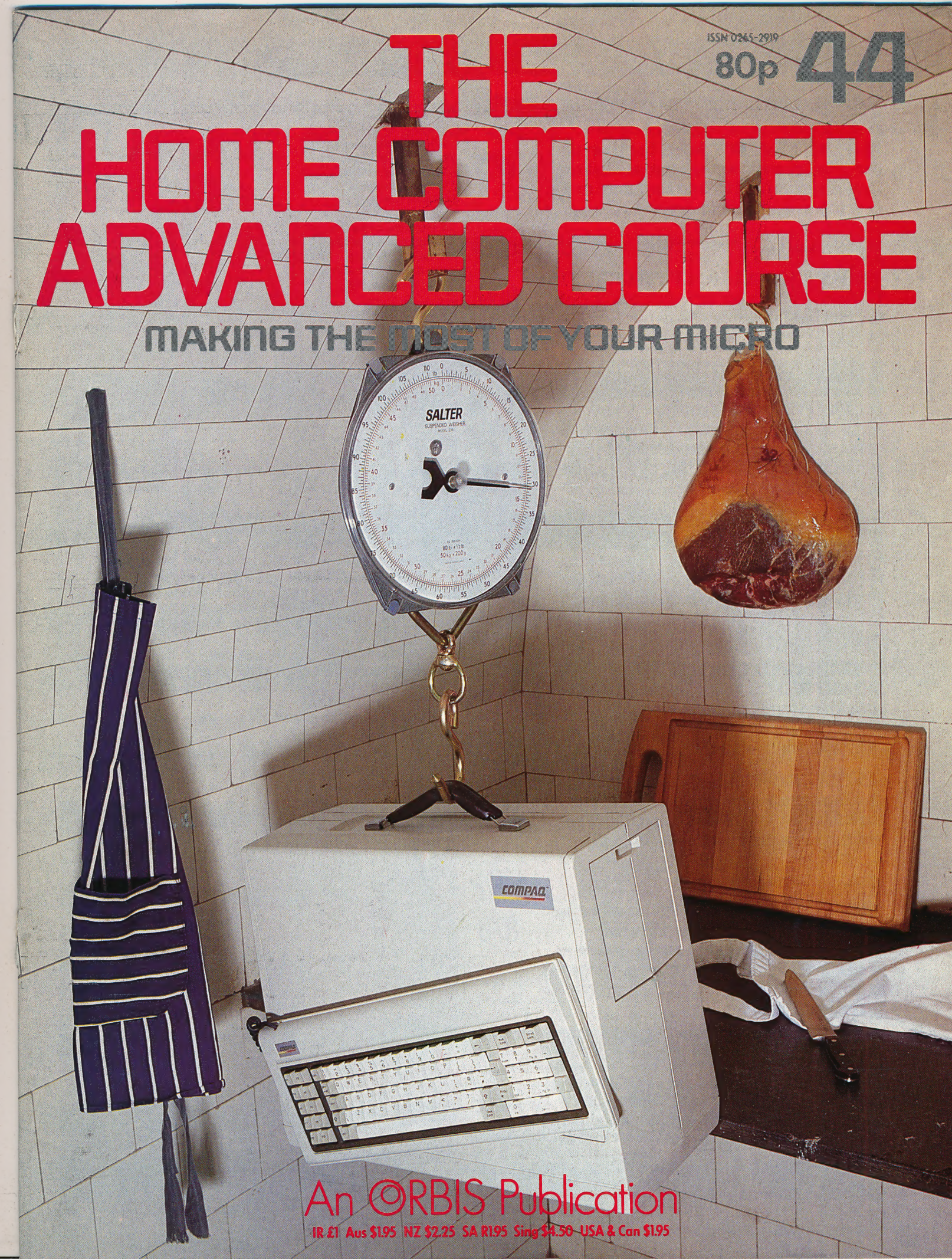
ISSN 0265-2919

80p 44

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **As the European market for computers expands, we look at the resultant problems involved in translating software from English.**
- **Research Machines has recently launched a new machine aimed at the educational market. We take a look at the Link 480Z.**
- **Continuing our look at vertical software, we examine some custom-built spreadsheets.**

DON'T VAT THE PRESS

There are strong reasons to believe the Chancellor of the Exchequer is planning to impose VAT on your magazine.
Such a move would turn the clock back 130 years — the last tax on newspapers and journals was repealed in 1855. Since then 'No tax on knowledge' has been a principle agreed by all Governments, even in the darkest days of war.
A free Press is a tax-free Press.
No Government should be given the power to impose financial pressure on a Press it may not like.
Tell your MP to say 'NO' to any tax on reading.

Issued by the Periodical Publishers Association, London

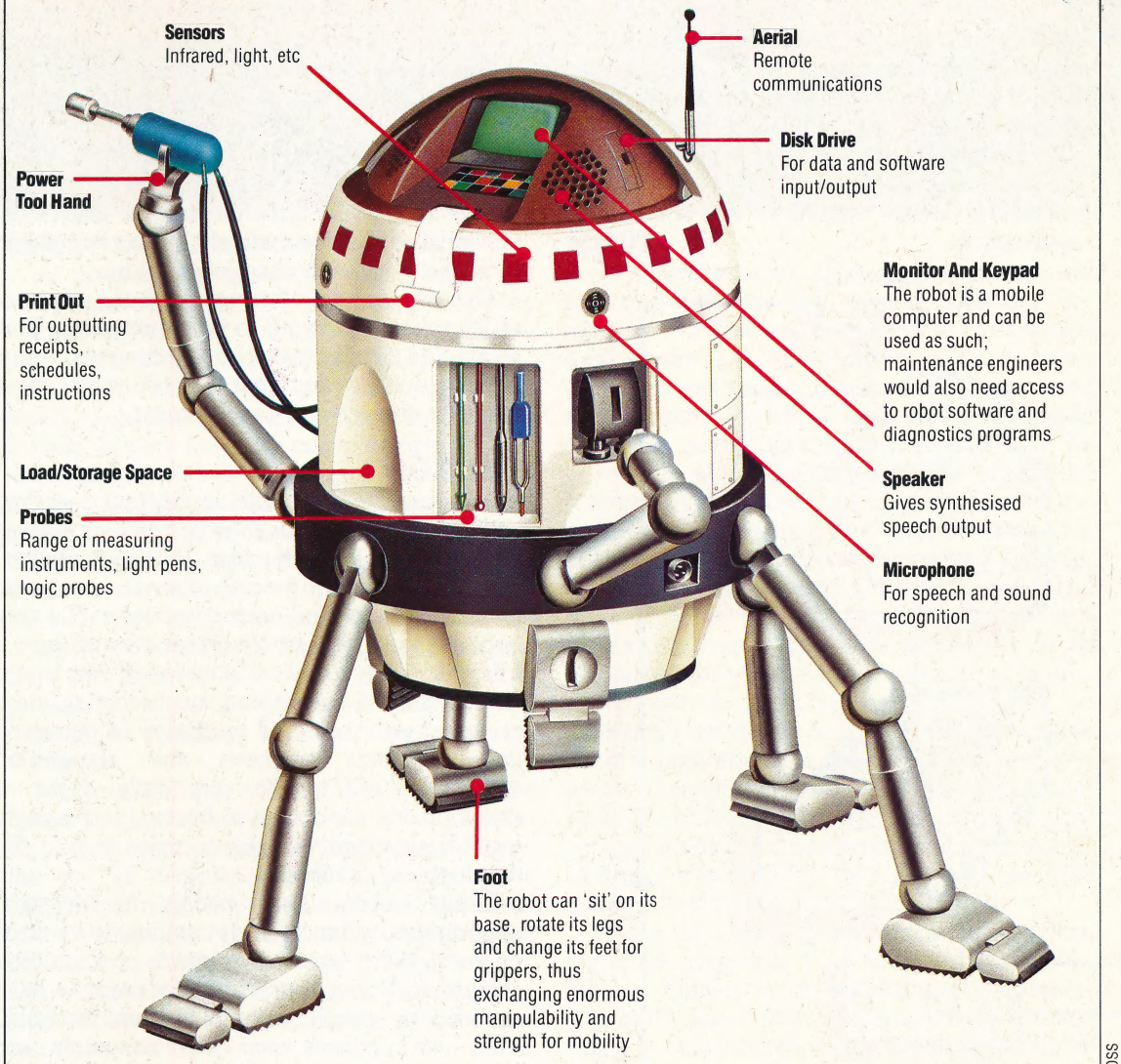
Editor Mike Wesley; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Sub Editor** Steve Mann; **Consultant Editor** Steve Colwill; **Contributors** Geoff Bains, Harvey Mellar, Karl Dallas, Joe Pritchard, Steve Colwill, Chris Naylor; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Controller** Peter Taylor-Medhurst; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984; Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 80p/IR£1. Subscription: 6 months: £23.92. 1 Year: £47.84. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 80p. Subscription: 6 months air: £37.96. Surface: £31.46. 1 year air: £75.92. Surface: £62.92. Binder: £5.00. Airmail: £5.00. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 80p. Subscription: 6 months air: £43.94. Surface: £31.46. 1 year air: £87.88. Surface: £62.92. Binder: £5.00. Airmail: £8.30. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/80p. Subscription: 6 months air: £51.74. Surface: £31.46. 1 year air: £103.48. Surface: £62.92. Binder: £5.00. Airmail: £9.45. **SOUTH AFRICA** – Price: SA R1.95. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 80p. Subscription: 6 months air: £55.38. Surface: £31.46. 1 year air £110.76. Surface: £62.92. Binder: £5.00. Airmail: £9.85. **AUSTRALIA** – Price: Aus$1.95. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZ$2.25. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 6711. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH HAM COURTESY 'PASTA PLACE', HEATH STREET NW3

FUTURE PERFECT

Metal Collar Worker



STEVE CROSS

Future Tense
If the general-purpose human-like robot is ever developed at a cost that makes it a reasonable substitute for semi-skilled human labour, it would need a highly developed 'intelligence' comprising knowledge database, sensory integration, skills database and learning software. An intelligence like this could be packaged in a variety of bodies — we show one possible type that might function as a semi-skilled light or heavy industrial worker

Our robotic series has concentrated in some detail on all the various aspects of robot behaviour. In this concluding instalment we consider the practical limitations imposed on present-day robot design and discuss possible future developments in the field as a result of new technological advances.

Our robotics series has shown how the real world of robots remains far removed from the fictional concept of mechanical thinking beings. Our own imaginations have conditioned us to expect certain things of robots. We expect them to be able to move around freely, under their own power; to see, hear, and feel the world around them; to converse with us on philosophy and science, or at least to communicate with us in an intelligent fashion; and to manipulate objects and ideas as we would do. We have, in other words, created robots in our own image. When we look critically at existing commercial, industrial, and hobbyist robots, we are often surprised at how well they can accomplish their specific tasks, while still feeling disappointed that they cannot do more.

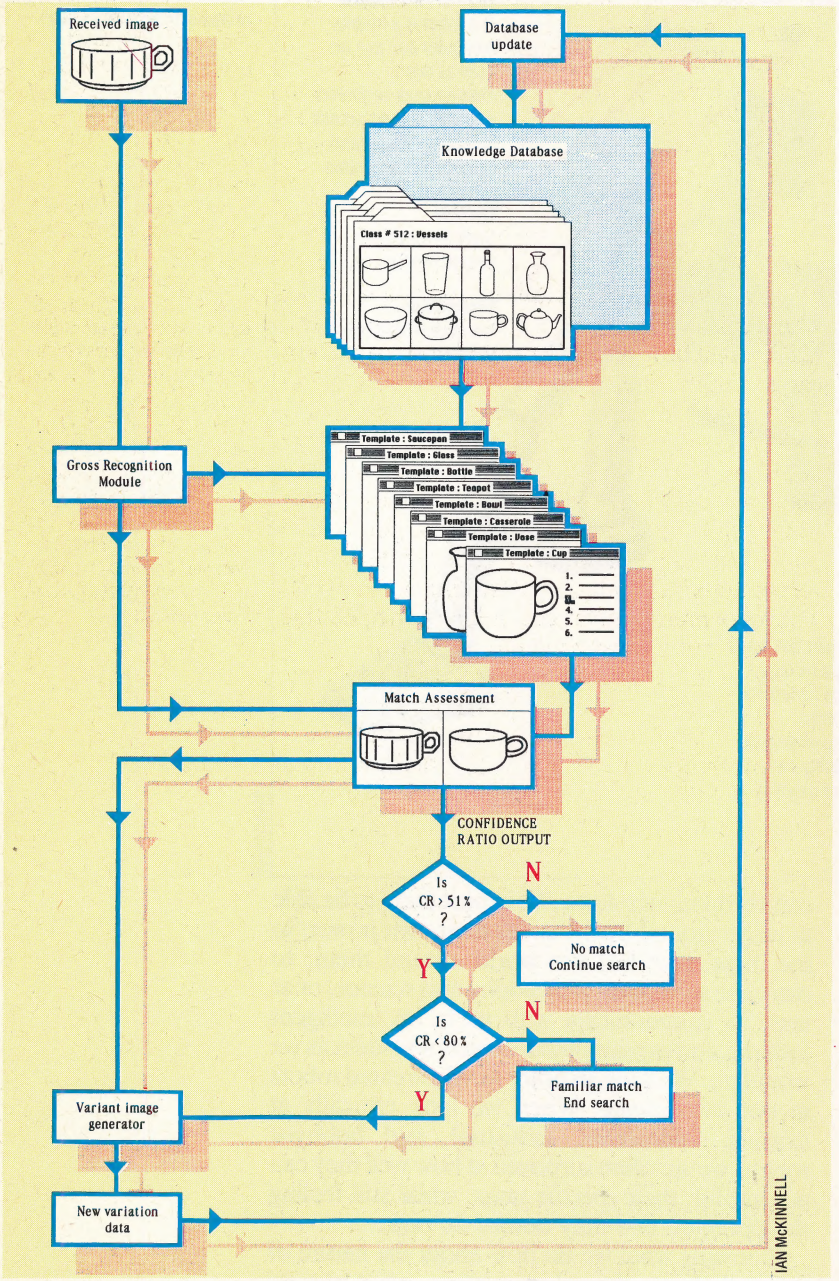
Knowing what we do now about the nature of robot design and implementation, what can we

Visual Variations
Knowledge of an object can be stored as a 'template' image of the archetypal object, plus a series of variation data statements; each variation statement can be applied to the template to produce a different image that still fits the type definition. An image from the robot's sensors is scanned by some gross analysis module that gives a first guess at the class of object in view. Each object in that class is then matched in all its variations against the received image until a match is found. The statistical confidence with which this match is made determines whether the new image's variations from the template are radical enough to require a new variation data statement to be generated from it

expect, realistically and practically, of future robots? Let's take a look at each of the major design areas in turn.

MOVEMENT

It is extremely unlikely that robots will walk on anything resembling a human leg in the near future. Too much processing time and space would have to be devoted to the effort of maintaining balance, while robot joints and electric or hydraulic musculature lack the flexiblity or freedom humans are accorded by the interaction of muscle, tendon, and cartilage. In addition, there are many times when a robot would be severely hindered by having to get around on two legs. Recently, though, some experimental robots have been built with four or six legs, resembling insects. This may offer an interesting design variation for some robot applications.



For other applications, such as military operations, planetary exploration, and more conventional uses around the home, wheels currently offer the most practical method of movement, and this is unlikely to change. Robot movement will become more fluid, but will probably never rival the beauty and grace of a human athlete in motion.

Many industrial robots and smaller robot arms are necessarily fixed in place, with their movements confined to a very specific area of action, since they are designed to perform one or two well-defined tasks. Unless the nature of industrial assembly changes radically, even roving industrial robots will probably remain relatively confined: they will continue to follow tracks, ride on rails, or swing from overhead racks. It is possible that advances in automation and robotic design will bring about a radical change in industrial production methods, but it is impossible to predict what such changes might be.

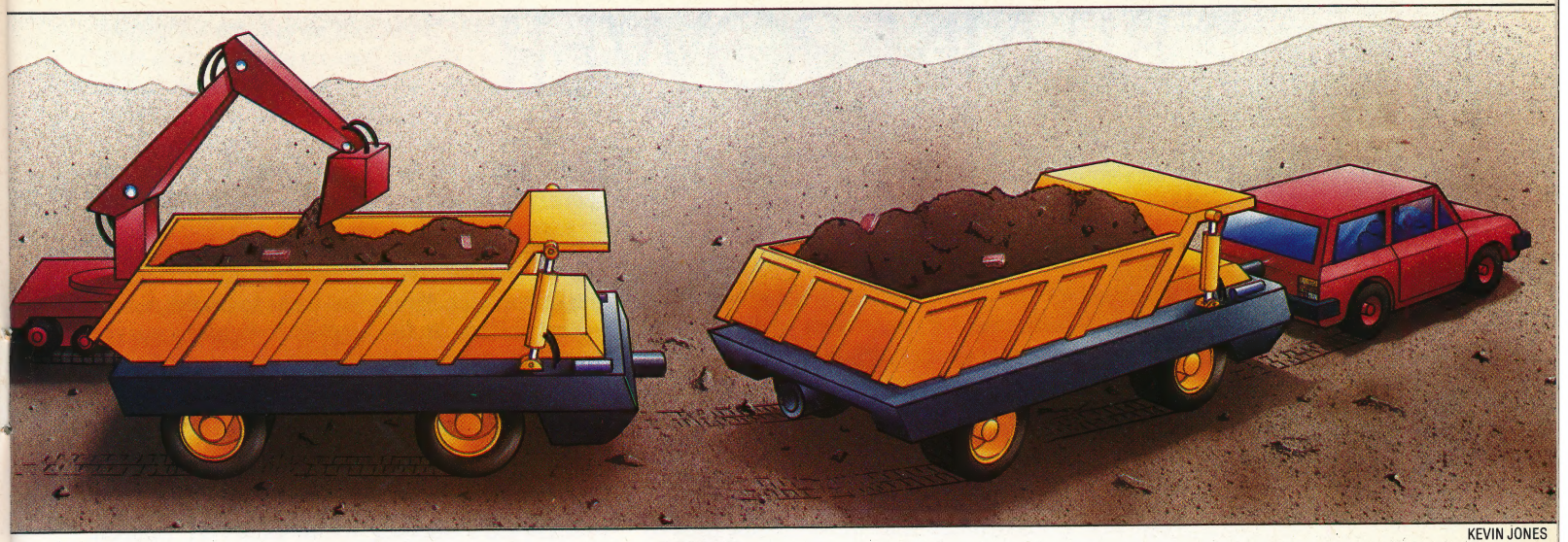
One key element of movement is the necessity of the robot to respond to its environment. This means that the robot must be well equipped with a sensory system, and that sensory input must correlate directly with its movement.

SENSORS

Robots can be fitted with sophisticated sensing equipment that extends their perceptions in areas new or unfamiliar to humans. Proximity sensors, motion detectors, precise discrete positional feedback devices and noise detectors with a very large range of perceptible frequencies all give a robot the ability to collect more varied data than a human can. Visual systems are becoming more accurate, with increased resolution of perceived images. Voice synthesis and recognition techniques, still in the very early stages of development, are certain to become increasingly sophisticated and will play a major part in the development of robotics.

Utility robots, used mainly in industrial applications, will most likely continue to be fitted only with those sensors required to perform their given tasks. Robot arm welders, for example, have no need for speech or complex visual feedback. They can perform their jobs accurately and quickly, with a minimum of sensory input, and extraneous perceptions would possibly be a hindrance.

General-purpose robots, designed to learn from experience and emulate human thought processes, would have to be fitted with as many sensors as possible. It would be crucial for the robot to be able to investigate its environment independently and assimilate the information it gathered. Humans rely heavily, for instance, on a combination of visual and auditory feedback to comprehend speech. We often tend to ignore this synthesis of sensations, particularly when thinking in terms of robot design. But for a robot to communicate with a human being, to understand language rather than merely recognise it, such a



KEVIN JONES

Convoy !
The techniques of robotics are likely to make their biggest impact on society when incorporated in special-purpose low-grade tools such as cranes, earth-movers, local delivery vehicles and heavy transport. Here we show a robot earth-mover loading a train of robot trucks on a construction site. When loaded, each truck moves semi-intelligently to an assembly point and hitches itself to a train of trucks. The train moves along the public roads under the power of the individual trucks that comprise it, but is controlled by the human driver of the lead vehicle. Blending the human skills of decision and command with the brute strength and single-minded intelligence of robots is likely to prove the cheapest and most profitable use of existing resources and future technology

union of sight and sound would be crucial.

At present, the amount of data a robot can accept and deal with is severely restricted by the amount of memory required to store sensory input, and by limitations on processing power and speed. A robot can store a visual image of an object, such as an apple, and connect the image to a name. Storing the image occupies memory, and the better the robot's visual resolution the more memory is needed to maintain the image. Since all apples are not alike, however, the robot must either have enough memory to store a characteristic sampling of apple images, or an algorithm that recognises variations and is able to rotate the basic image so the apple can be viewed from any position. With even a minimal level of resolution — say 256 pixels per image — the number of variations can reach well into the thousands.

Memory demands will probably be met in the future by higher capacity RAM chips (1 Mbit chips are currently being developed), and through the use of dedicated RAM chips that store 'variational' data. The general-purpose data held in these chips can be called on as needed by the processor to clarify a variety of different images, almost like a subroutine in a BASIC program.

In addition to a huge memory, true sensory awareness for a robot would require data to be coming from many sources simultaneously, and to be processed very rapidly. Existing processors would be unable to handle the sheer volume of information coming in at any one time, and data would soon begin to pile up, waiting to be processed. The likely solution to this problem is the use of two or more high-speed, high-capacity processors, working in parallel. A controlling processor could then act as a manager, distributing tasks to idle processors elsewhere in the system.

Progress is being made very rapidly toward solving the hardware problems facing researchers. But a robot will need very complex software to enable it to understand what is being processed. In other words, the robot needs a mind to know what to do with its perceptions.

MIND

As we have seen in discussing movement and sensors, there are two major directions for robotics. The first, and most likely to be exploited soon, is the area of intelligent tools, or utility robots. Industrial arms and automated manufacturing systems that perform specific tasks, no matter how intricate, need only be provided with carefully-defined controlling software. A robot arm can be given a set of co-ordinates and programmed to execute a sequence of actions without understanding what it is doing, where it is or any details of its environment. The result might be a perfectly painted door panel on a car assembly line, or a well-cleaned car in an automated car wash. As robots are asked to do a wider variety of jobs, however, these are necessarily less clearly defined. If they must move about a room where the contents change from day to day, they must be able not only to gather and process information, but also to incorporate new perceptions into their understanding of the world. The robot must be given controlling and operating software, but it must have room to grow.

Trying to create a mechanical mind opens important questions, as yet to be resolved, about the way humans think and learn. For instance, how much does a human infant actually know at birth? Is a human adult entirely the product of its environment, or of its heredity, and what is the relationship between the two? Does a human being start out with a set of internal constructs that help it learn language and mathematics and aesthetics, and if so how do they work? As Igor Aleksander and Piers Burnet point out in their robotics book, *Reinventing Man* (published by Kogan Page in 1983), it is difficult to answer these questions without being able to experiment directly on the human brain. Perhaps, in the future, the robots we create in our image will help us to improve our understanding of ourselves. Although such questions are not purely theoretical, as experiments being carried out now aim to accomplish just this, the idea of a thinking robot is unlikely to be realised for a very long time.

BRAIN POWER

In the introduction to our vertical software series (see page 844), we mentioned how Caxton Software's BrainStorm program was being used by a group of parents of teenage heroin addicts to organise their campaign against hard drugs. Here, we take a detailed look at this unique package.

BrainStorm has been called a 'thought processor', but that doesn't mean that the program tries to supersede the brain. What BrainStorm really does is help you organise your thoughts. A direct analogy with pre-electronic methods is useful in understanding precisely what it does.

In planning any complex project, it helps to make lists of what you have to do. A list of the main objectives will generate sub-lists of how to complete each item on the preceding list, and so on until the planner gets dizzy from shuffling around pieces of paper. Any changes — deciding that a main list item should be in the sub-list of another item, for instance — can mean so much erasing and writing-in that the system becomes unmanageable. BrainStorm makes such changes and developments as simple as the cut-and-paste functions found on word processors — so the expression 'thought processor' isn't quite so misleading, after all.

Any item in each list or sub-list can be *promoted* into becoming the heading for a lower sub-list, and if similar activities in different lists need to be linked together, they can be given the same name, termed *namesakes*. In that case, anything added to a sub-list headed by a namesake is added to all the other lists using the namesake.

Anyone familiar with the rather unfriendly editing commands of a word processing package such as WordStar will find it quite easy to get used to the non-mnemonic cursor control commands. Pressing the Control key (CTRL) in conjunction with the S key moves the cursor to the left. CTRL — D, CTRL — E and CTRL — X move the cursor right, up and down respectively, and these four keys make a logical pattern on the keyboard. In any case, it is possible to redefine these. Less easy to get used to is the fact that you can cursor up and down when entering text, but have to change to 'amend' mode (using CTRL — A) to move left or right within a line.

PROGRAM OPERATION

BrainStorm's opening menu offers 11 options, each selected by an initial letter, most of which are self-explanatory: Use, Load, Print, ID Drive (to log-on to a different drive from the default), Clear (to erase the current model from memory), Save, Write (to print to disk), Directory, Xit, Merge, and Kill.

To begin, the user presses U, and the program enters typing mode immediately. Ideas can be entered as a more or less random list (known as a *model*). For instance, an example model might consist of:

```
Read manual
Start typing list
Type sub-list
Edit list
```

The control commands are not revealed immediately, but will be listed by pressing ?. By moving the cursor up to any item in the list and pressing CTRL — R, the designated item is 'promoted' to being the heading of a sub-list. Similarly, any item on that sub-list can be made the heading of a new sub-list, and so on until memory runs out. To get back to the previous list, CTRL — C is pressed.

Items can be moved around, either within lists or to lists on other levels, by labelling them with the @ sign, and then using CTRL — G (for Get) or CTRL — P (for Put) to execute the move. After CTRL — G is used, the @ sign automatically moves to the next item on the list; therefore, pressing CTRL — G again will get the next item, and so on. This is a valuable facility for moving a whole series of items to a lower or higher sub-list.

If you wish to insert new items within an existing list, all you have to do is to move the cursor to the beginning of the line that is to follow the new item, and type it in. When you press RETURN the rest of the list is moved down a line to make space for it. Pressing CTRL — A (for Amend) allows any item to be altered.

By giving items in different lists the same name — for example, a date — they become *namesakes* and are automatically cross-referenced. So if a given list required a certain event to happen on a

particular day, say on January 1, a list of events for that date could be created, and this would allow a schedule of the day's activities to be continually updated by cross-referencing.

Any number of namesakes can be created, and then accessed in sequence, using CTRL — S for the next and CTRL — D for the previous occurrence. This is called a *duckshoot* sequence, so that after the last occurrence, CTRL — S will loop back to the first.

PRINTING LISTS

It is, of course, possible to print out the lists, and they are usually formatted with the indentation signifying the levels of the list. For example:

```
Read manual
    Take manual from box
        Turn to contents
            Find page required
        Read page
    Return manual to box
Start typing list
    Press CTRL — R to promote list item as new
    heading
        Type sub-list
    Press CTRL — C to return to previous list
Edit list
    Press CTRL — A to alter an entry
```

The amount of indentation is specified by the user.

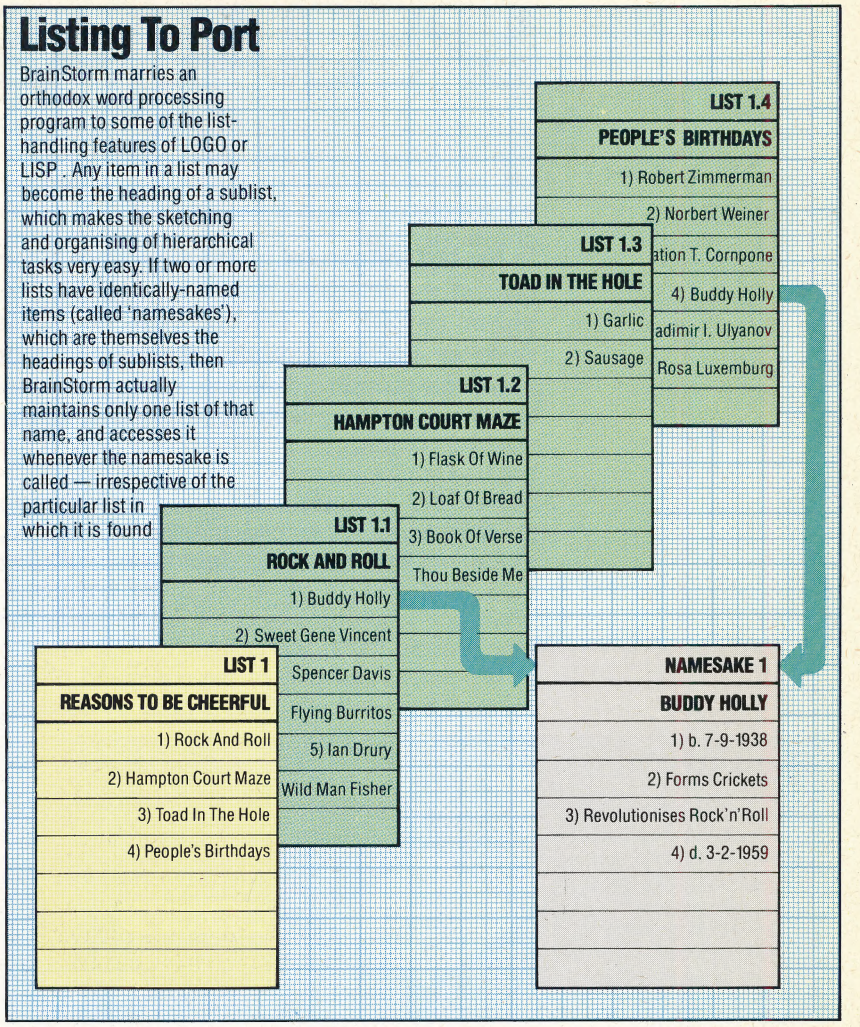
It is also possible to edit lists outside BrainStorm. If they have been saved using the Write-to-disk command, they will be recognised as documents by WordStar and other word processing programs, and can be edited, printed and, if necessary, reSAVEd. This means, for instance, that BrainStorm could be used for preparing the synopsis of a book, which could then be written using a regular word processing program, accessing the BrainStorm.Doc file as the book progresses.

Because of the flexibility of the program, this process can start with the very earliest jotting down of rough ideas — again, something that is normally done on scraps of paper — which then become chapter headings (under which the contents are listed). If an item within a chapter looms large enough to become a chapter in its own right, this can easily be accomplished. Similarly, chapter headings that turn out to be less important than was thought at first can be 'demoted'.

Thus, the rough outline of the book begins to reveal itself, and since WordStar or similar programs can access BrainStorm files, the transformation of this outline into the finished manuscript flows naturally out of the earliest thought processes. It also means that it is not necessary to adopt the rather laborious procedures necessary within BrainStorm, for instance, if one wishes to print in double or triple spacing. Douglas Adams, the author of *The Hitch-hiker's Guide to the Galaxy*, used BrainStorm to develop the adventure game based on his books using this method.

Listing To Port

BrainStorm marries an orthodox word processing program to some of the list-handling features of LOGO or LISP. Any item in a list may become the heading of a sublist, which makes the sketching and organising of hierarchical tasks very easy. If two or more lists have identically-named items (called 'namesakes'), which are themselves the headings of sublists, then BrainStorm actually maintains only one list of that name, and accesses it whenever the namesake is called — irrespective of the particular list in which it is found



The program comes with a sample model (called SAMPLE.BRN), which includes the skeleton of a schedule planner, name-and-address file, a list of tasks (sub-listed as 'urgent', 'important', and 'don't forget'), plus a notebook section. This is quite valuable, and can be added to one's own models using the Merge option.

BrainStorm is easy to learn to use. The loose-leaf manual is very clear (it was written using BrainStorm itself), but the commands are always available from an on-screen menu, making constant reference to the manual unnecessary.

Not all the commands are mnemonic, so some may be hard for users to remember, but a program called INSTALLB is provided, which allows every command to be reconfigured, and the menus to be altered to conform to the new set-up. This is extremely clearly set out, and is completely menu-driven.

BrainStorm: For about 25 CP/M, MS-DOS and PC-DOS machines, including IBM, Sirius and Apricot
Price: £339.25
Publishers: Caxton Software Ltd, 10-14 Bedford Street, London WC2E 9HE
Authors: David Tebbutt and Mike Liardet
Format: Disk

SPECIAL ASSIGNMENT

In the last instalment of our adventure game project we designed routines to enable the player to pick up objects. Now we must develop the corresponding routines that allow the player to drop any objects he may be carrying. We also look at the first of the 'special' locations.

The DROP subroutine bears many similarities to the TAKE routine described on page 846. Indeed, we can use the same object checking routines that were developed for use with the TAKE command. Three checks on the object are made during the TAKE routine. The first is designed to test whether or not the second part of the command phrase contains a valid object. This is done by checking each word of the command phrase systematically against the object names in the inventory array — IV$(,). If a match is found then a variable, F, is set, giving the position of the matched object within the array. This validity check must also be used in the DROP routine to establish whether the object exists and, if it does, to determine its position in the inventory.

The second check used in the TAKE routine is also used in the DROP routine; this tests whether the player holds the object specified in the command in the inventory of carried objects — IC$(). Obviously, a player cannot drop an object that he is not carrying! The third test used in the TAKE routine checks to ensure that the object to be picked up is at the player's current location, as determined by the position variable, P. However, as the object to be dropped must be held by the player, its position will not appear in the main inventory, and this third test is, therefore, not needed by the DROP routine.

Assuming that both tests result in a favourable outcome, then the following changes must be made to both the main and the player's inventories:

1) The position of the object to be dropped will now be specified by F. The current position, P, must be entered in the main inventory array in position IV$(F,2).
2) The object description must be deleted from the player's personal inventory of objects carried, IC$(). This is best done by searching through the array until the appropriate object is found and replacing it with a null string.

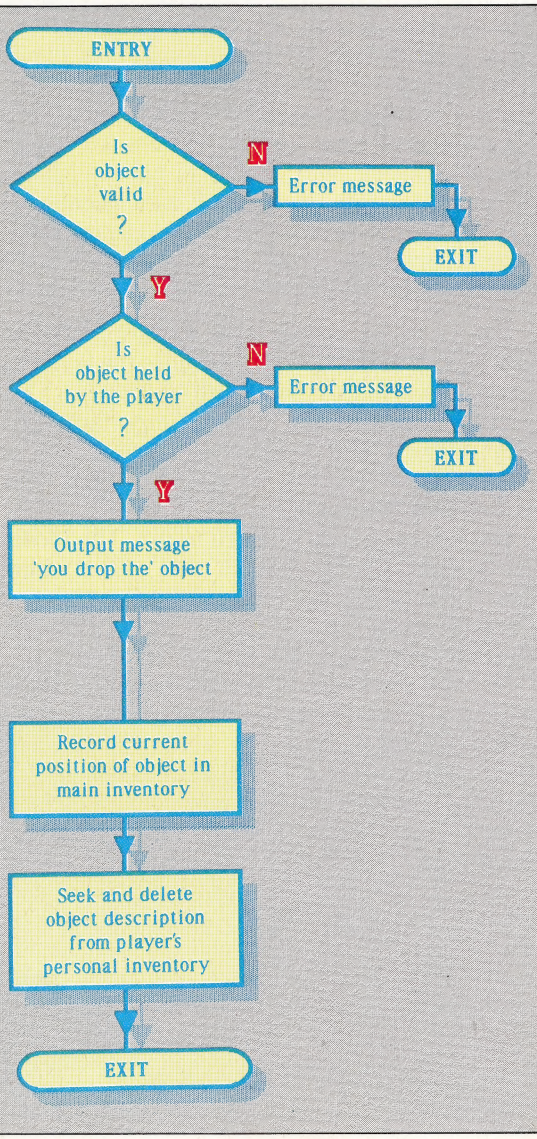
The logic of the DROP routine is shown in the flowchart. Here is the listing for the routine in the Haunted Forest game:

```
3900 REM **** DROP S/R ****
3910 GOSUB5300:REM VALID OBJECT
3920 IF F=0 THEN SN$="THERE IS NO "+W$:GOSUB5500:
RETURN
3930 :
3940 REM ** IS OBJECT IN CARRIED INVENTORY **
3950 OV=F:GOSUB5450
3960 IF HF=0 THEN SN$="YOU DO NOT HAVE THE "+IV$
(F,1):GOSUB5500:RETURN
3970 :
3980 REM ** DROP OBJECT **
3990 SN$="YOU DROP THE "+IV$(F,1):GOSUB5500
4000 IV$(F,2)=STR$(P):REM MAKE ENTRY IN INVENTORY
4010 :
4020 REM ** DELETE OBJECT FROM CARRIED INVENTORY
**
4030 FOR J=1TO2
4040 IF IC$(J)=IV$(F,1) THEN IC$(J)="":J=2
4050 NEXT J
4060 RETURN
```

It can be seen that one of the major advantages of programming in modules is that the same routines can be accessed for different purposes. By using a system of flags, decisions can be made within short

Drop It!

The logic of the DROP routine is similar to that of the TAKE routine (see page 846), but only the validity of the object and its presence in the player's inventory need be checked before the DROP command is executed. The main inventory is then updated to show a new location for the object, and the object name is deleted from the player's inventory



IAN McKINNELL

subroutines that are not acted upon until control is returned to the routine that called the subroutine. A good example of the use of this type of program structure is the validity test described earlier. This subroutine is called by both the TAKE and DROP routines. In each case, the subroutine makes a decision as to the validity of the object part of the command phrase. However, the flow of the program is not altered until a RETURN is effected to either the TAKE or DROP routines. Only after returning is the value of the flag, F, set by the validity test subroutine and the appropriate branch made. One criticism of this technique is that we are effectively testing the same condition twice — once to set the flag value, and again to test the value of the flag. Although this is true, the added flexibility and ease of debugging achieved by employing this technique usually outweighs the slightly longer execution time that results.

SPECIAL LOCATIONS

We are now at the point in our project where we have completed the programming of the game's skeleton; that is, the programming that allows the player to carry objects and move around in the adventure world. We can now move on to the next phase of design in which we consider the 'special' locations where objects are put to use, perils are met and where the player's ingenuity and skill are tested.

Before we look in detail at the programming of the routines for one of the special locations in the Haunted Forest, let's consider the additions to be made to the main program loop in order to detect special locations. These two lines must be inserted into the listing:

```
257 GOSUB2700:REM IS P SPECIAL ?
258 IF SF=1 THEN 300:REM NEXT INSTRUCTION
```

Line 257 calls a subroutine to see if the current location is special. If this is the case then a 'special flag', SF, is set to one. This means that when control is eventually returned to the main program loop, the part of the main loop dealing with instructions can be avoided. The subroutine that decides whether the current location is special or not is:

```
2700 REM **** IS P SPECIAL S/R ****
2705 SF=0:REM UNSET SPECIAL FLAG

2716 REM ** OTHER SPECIAL LOCATIONS **
2720 ON P GOSUB4590,4690,4790,4590
2730 RETURN
```

You will recall that, when we designed the original map for the Haunted Forest, we numbered the four special locations first (see page 766). We can, therefore, simplify the selection of the appropriate subroutine for each special location by making use of the ON. . . GOSUB command. As can be seen by the way it is used in line 2720, this command is followed by a series of line numbers, and the appropriate line number is selected according to the value of P. If P is one, for example, the command will GOSUB to the first line number from the list; if P is two, then the second line number will be used for the GOSUB call, and so on.

There are four line numbers, one for each of the special locations in Haunted Forest. If P exceeds four, then control simply passes to the following line. If each of the four subroutines that can be called from line 2720 sets an SF flag, then the fact that P was a special location can be flagged. If no routine is called, the SF flag will remain set at zero, indicating that P is just an ordinary location. The ON. . . GOSUB command is clearly an economical alternative to a series of IF. . . THEN statements testing the value of a variable and branching to different subroutines accordingly.

THE TUNNEL ENTRANCE

Two of the special locations in the Haunted Forest are the two entrances to a tunnel (locations 1 and 4). To deal with the simple scenario of the player wishing to enter the tunnel, we need to construct carefully a routine that handles the normal commands and allows the player to enter the tunnel or retreat back down the path.

```
4590 REM **** TUNNEL ENTRANCE S/R ****
4600 SF=1
4605 SN$="YOU HAVE ARRIVED AT THE MOUTH OF A LARGE
TUNNEL":GOSUB5500
4610 SN$="YOU CAN ENTER THE TUNNEL OR RETREAT
ALONG THE PATH":GOSUB5500
4620 :
4625 PRINT:INPUT"INSTRUCTIONS";IS$
4630 GOSUB2500:REM SPLIT INSTRUCTION
4635 IF F=0 THEN 4625:REM INVALID INSTRUCTION
4637 GOSUB3000:REM NORMAL INSTRUCTIONS
4640 IF MF=1 THEN RETURN:REM PLAYER RETREATS
4645 IF VF=1 THEN 4625:REM INSTRUCTION OBEYED
4650 REM ** NEW INSTRUCTIONS **
4655 IF VB$="ENTER" THEN GOSUB 4700:RETURN
4660 IF VB$="RETREAT" AND P=4 THEN MF=1:P=6:RETURN
4665 IF VB$="RETREAT" AND P=1 THEN MF=1:P=9:RETURN
4667 SN$="I DON'T UNDERSTAND":GOSUB5500:GOTO 4625
```

The routine starts by setting SF to one to indicate that a special location has been reached. After displaying a message on the screen, describing the tunnel entrance and the options open to the player, an instruction is asked for. Once again, rather than re-inventing the wheel each time we wish to analyse an instruction, we can take advantage of the modular construction of the program to call up the 'split instruction' and 'normal command' subroutines developed for use in the TAKE and DROP routines. By considering carefully the states of the various flags set by these two subroutines, we can transfer control within our new routine as required. Let's consider these flags individually.

The F flag set by the 'split instruction' routine indicates whether the instruction passed to it has a valid format. If the instruction is a one-word command not recognised by the routine, then F takes the value zero — in which case we will want to loop back to get another instruction.

The MF flag is set by the 'normal command' routine if a description of a location is required — this happens when a GO or LOOK command is issued. A RETURN to the main program loop will allow the new location to be moved to, in the former case, or the same location to be described and the special routine re-entered, in the latter case.

The VF flag is also set by the 'normal command' routine. A value of one indicates that the

instruction was recognised and obeyed, in which case we should loop back for the next instruction. If VF<>1 then the command is not one of the normal commands. Having dealt with the normal command possibilities we can add new commands to this routine. In this case, two such instructions are included: ENTER, to go into the tunnel, and RETREAT, to move one location away from the tunnel entrance. As this routine is designed to work for both entrances to the tunnel, the RETREAT command must take account of which end of the tunnel the player is negotiating — this is indicated by P taking the value 1 or 4. P can, therefore, be reset accordingly before leaving the routine so that a change of location is made on re-entry to the main program loop.

The special perils that await the adventure player once inside the tunnel are the subject of the next instalment.

Digitaya Listings

```
1190 GOSUB2670:REM IS P SPECIAL
1200 IF SF=1 THEN 1250:REM NEXT LOOP

2360 REM ** DROP S/R **
2370 GOSUB5730:REM IS OBJECT VALID
2380 IF F=0 THEN PRINT"THERE IS NO ";W$:RETURN
2390 :
2400 REM ** IS OBJECT HELD ? **
2410 OV=F:GOSUB5830
2420 IFHF=0THENPRINT"YOU DO NOT HAVE THE ";IV$(F,1
):RETURN
2430 :
2440 REM ** DROP OBJECT **
2450 SN$="YOU DROP THE "+IV$(F,1):GOSUB5880
2460 IV$(F,2)=STR$(P):REM UPDATE OBJ POSITION
2470 :
2480 REM ** DELETE FROM HELD OBJ LIST **
2490 FORJ=1TO4
2500 IF IC$(J)=IV$(F,1)THENIC$(J)="":J=4
2510 NEXTJ
2520 RETURN

2670 REM **** IS P SPECIAL S/R ****
2680 SF=0:REM UNSET SPECIAL FLAG

2710 ON P GOSUB 2850,2960,3450,3830,4180,4550,5150
2720 RETURN

2850 REM **** TV OUTLET S/R ****
2860 SF=1
2870 SN$="YOU HAVE ENTERED THE TV OUTLET AND THERE
 IS NO ESCAPE."
2880 SN$=SN$+"YOU ARE DOOMED FOREVER TO BE A TV CH
AT SHOW HOST"
2890 GOSUB 5880:REM FORMAT PRINT
2900 PRINT
2910 PRINT"WELCOME TO THE SHOW......
2920 FORJ=1TO500:NEXTJ
2930 GOTO 2910
2940 END

3830 REM **** JOYSTICK PORT ****
3840 SF=1
3850 SN$="A USER WITH RED-RIMMED EYES ZAPS HIS LAS
ER AT YOU REPEATEDLY."

3860 GOSUB5880:REM FORMAT
3870 :
3880 REM ** INSTRUCTIONS **
3890 RD=RND(TI):IF RD>.65THEN 4110:REM HIT
3900 PRINT:INPUT"INSTRUCTIONS";IS$
3910 GOSUB1700:GOSUB1900:REM ANALYSE INSTRUCTION
3920 IFMF=1THENMF=0:PRINT"YOU CAN'T MOVE...YET":GO
TO3880
3930 IFVF=1THEN3880:REM NEXT INSTRUCTION
3940 IFVB$<>"USE"THENPRINT"I DON'T UNDERSTAND":GOT
O3880
3950 GOSUB5730:REM IS OBJECT VALID
3960 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO3880:REM
 NEXT INSTRUCTION
3970 :
3980 REM ** IS OBJECT LASER SHIELD **
3990 IF F=3 THEN4020:REM OK
4000 SN$="YOUR "+IV$(F,1)+" IS NO USE":GOSUB5880:G
OTO3880
4010 :
4020 OV=3:GOSUB5830:REM IS LASER SHIELD CARRIED
4030 IFHF=0THENSN$="YOU DO NOT HAVE THE "+IV$(3,1)
:GOSUB5880:GOTO3880
4040 :
4050 REM ** SAVED **
4060 SN$="YOU USE THE LASER SHIELD TO PROTECT YOUR
SELF. A BLAST KNOCKS"
4070 SN$=SN$+" YOU OUT OF THE JOYSTICK PORT AND BA
CK INTO THE MACHINE."
4080 GOSUB5880:REM FORMAT
4090 P=INT(RND(TI)*40+7):MF=1:RETURN
4100 :
4110 REM ** HIT **
4120 SN$="YOU ARE HIT BY THE LASER AND YOU ARE ONL
Y DIMLY AWARE THAT"
4130 SN$=SN$+" YOUR ATOMS HAVE BEEN DISTRIBUTED TO
 THE FOUR CORNERS"
4140 SN$=SN$+" OF THE UNIVERSE"
4150 GOSUB5880:REM FORMAT
4160 END

5150 REM **** GATEWAY TO MEMORY S/R ****
5160 SF=1
5170 SN$="AN USHER GREETS YOU BUT TELLS YOU THAT Y
OU CANNOT BE ADMITTED"
5180 SN$=SN$+" UNLESS YOU GIVE AN ADDRESS":GOSUB58
80
5190 REM ** INSTRUCTIONS **
5200 PRINT:INPUT"INSTRUCTIONS";IS$
5210 GOSUB1700:GOSUB1900:REM ANALYSE
5220 IF MF=1 THEN RETURN:REM MOVE OUT
5230 IF VF=1 THEN 5200:REM NEXT INSTRUCTION
5240 IF VB$<>"GIVE"THENPRINT"I DON'T UNDERSTAND":G
OTO 5200
5250 :
5260 GOSUB5730:REM IS OBJECT VALID
5270 IFF=0THENPRINT"THERE IS NO ";W$:GOTO5200:REM
NEXT INSTRUCTION
5280 :
5290 REM ** IS OBJECT ADDRESS **
5300 IF F=1 THEN5330:REM OK
5310 PRINT"HE NEEDS YOUR ADDRESS":GOTO5200
5320 :
5330 OV=1:GOSUB5830:REM IS ADDRESS CARRIED
5340 IF HF=1 THEN 5370
5350 SN$="YOU DON'T HAVE THE "+IV$(1,1):GOSUB5880:
GOTO5200
5360 :
5370 REM ** OK PASS THROUGH **
5380 SN$="THE USHER LOOKS AT YOUR ADDRESS AND ALLO
WS YOU TO PASS"
5390 SN$=SN$+" THROUGH":GOSUB5880
5400 P=40:MF=1:RETURN
```

Basic Flavours

Spectrum:

In both programs use these alternatives for the variable names: S$ for SN$, R$ for NN$, V$(,) for IV$(,), I$() for IC$(), T$ for IS$, B$ for VB$. Substitute the following lines in the Haunted Forest listing:

```
2720 IF P=1 THEN GOSUB4590
2722 IF P=2 THEN GOSUB4690
2724 IF P=3 THEN GOSUB4790
2726 IF P=4 THEN GOSUB4590
```

Substitute the following lines in the Digitaya listings:

```
2710 IF P=1 THEN GOSUB2850
2711 IF P=2 THEN GOSUB2960
2712 IF P=3 THEN GOSUB3450
2713 IF P=4 THEN GOSUB3830
2714 IF P=5 THEN GOSUB4180
2715 IF P=6 THEN GOSUB4550
2716 IF P=7 THEN GOSUB5150
3890 LET RD=RND(1)
4090 LET P=INT(RND(1)*40+7)
```

BBC Micro:

Substitute the following lines in the Digitaya listings:

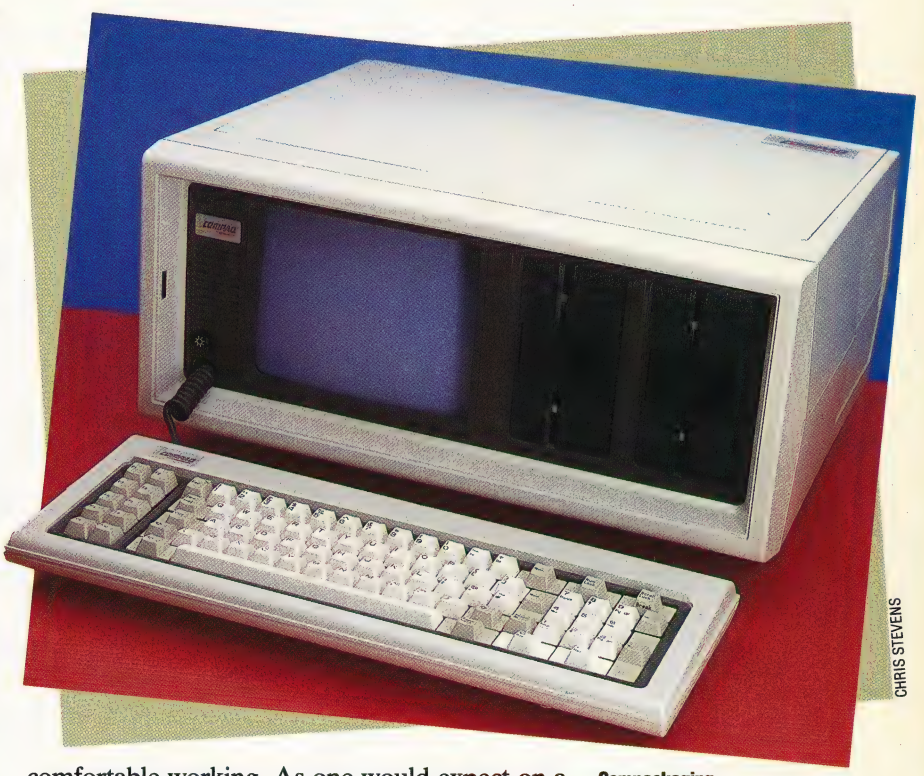
```
3890 RD=RND(1)
4090 P=RND(40)+7
```

PROFESSIONAL HEAVYWEIGHT

IBM-compatible machines are now dominating the business market. These are usually cheaper and offer more facilities than the machine they mimic, and many more are designed to be 'transportable'. Here we examine one of the earliest IBM-compatible portables, the Compaq Plus.

When the IBM Personal Computer was launched in the United States, it brought a certain respectability to the microcomputer. Businessmen, who had previously regarded such devices as little more than gimmicks, were happy to buy from the acknowledged giant of the computer industry. As software houses began to produce a vast number of programs for the IBM, it was perhaps inevitable that many hardware manufacturers would produce computers capable of running IBM software and so take advantage of the huge software base that was accumulating.

One of the first in this field was the Compaq Computer Corporation — a company formed specifically to produce an IBM-compatible portable machine, the Compaq Plus. The model examined here is the 256 Kbyte twin-disk version, although the machine is also available with a single drive, or with a 10 Mbyte fixed hard disk.



CHRIS STEVENS

Compackaging
The sleek and uncluttered lines of the Compaq's case are echoed in the simple front panel and its reasonable facsimile of the IBM PC keyboard. Flaps on either side of the case give access to the ports and power supply, and storage space for both the power lead and the mains plug — an important and often overlooked point!

CARRY THAT WEIGHT

'Portable' is perhaps something of a misnomer for a machine weighing 14kg (28lb) — it is certainly difficult to carry it for more than a short distance. However, Compaq has at least recognised the problem of moving such a large weight around and has provided a padded vinyl handle that makes carrying easier on the fingers.

As is usual with portable computers, the Compaq keyboard clips onto the front of the computer. As the carrying handle is fitted to the back, this means that the keyboard also serves as the base. However, the machine is sturdily built and the keyboard seems to accept the weight and subsequent rough handling quite happily. The complete unit is approximately the size of a sewing machine. Once unclipped, the keyboard is connected to the computer by a thick coiled cable. The manufacturer claims that this enables the keyboard to be positioned to allow comfortable working. However, once the keyboard is pulled more than about eight inches from the unit it begins to be dragged back by the coils, thus limiting the distance between screen and user. A thinner cable would have removed this restriction.

The keyboard, like the computer itself, has folding legs that enable it to be angled for more comfortable working. As one would expect on a business computer like the Compaq Plus, the keys are easy to use and reliable. The keyboard layout is identical to that of the IBM PC, with 10 function keys on the left-hand side of the typewriter keys and a numeric keypad on the right. The obvious advantage of this layout is that users who are familiar with the layout of the IBM will be able to use the Compaq keyboard instantly without having to learn the position of the new keys. The disadvantage is that the IBM keyboard is not ideal and therefore design problems are duplicated. The Enter key on the IBM is the same size as the other keys, which makes it difficult to find. Perhaps more seriously for touch-typists, the Shift key is not in its usual place below the 'A' key. Instead the backslash key occupies this position, which means a touch-typist will, for a while at least, be continually hitting backslash instead of Shift. These are problems that recur on the Compaq.

While running MS-DOS (the Microsoft version of PC-DOS), the function keys on the left of the keyboard act as editing aids. These functions change depending on the applications program being run. Under BASIC, for example, they become single keyword entry keys such as LOAD, SAVE and LIST. A similar system is also used for the function keys on MSX machines.

The numeric keypad on the right of the keyboard has a dual function. In normal operation

it can be used to move the cursor around the screen, but if the Num Lock key is pressed it can be used as a calculator.

The screen is a standard 17.7cm × 13.3cm green monitor, giving an 80 × 25 character resolution. The text is easily read and there is a brightness control on the right-hand side of the screen. On the left-hand side are the twin 5¼in floppy disk drives. As the standard configuration of the Compaq Plus is a single disk drive, the MS-DOS system master disk will assume this to be the case and will ask for all disks to be placed in drive A — this can be a nuisance when a disk is being copied as the user must continually swap disks. However, MS-DOS may be configured to make full use of both drives. The drives themselves appear to be quieter than their IBM counterparts.

OUTSIDE CONNECTIONS

Beneath a panel on the right-hand side of the machine are the interface ports. The Compaq Plus is fitted with a Centronics-type parallel printer port, an RGB interface and an RF port. Three expansion slots are also provided, allowing IBM plug-compatible boards to be fitted. Typical additions are extra memory boards, a VDU colour card or a modem that allows the computer to communicate over the telephone network. Beneath another panel on the left-hand side is the power input — a standard three-pin Eurosocket — above which is the on/off switch. To the side is the fan that keeps the inside of the machine cool. Inside the computer, the circuitry is protected by a metal casing (which accounts for the machine's weight); this not only provides protection against rough handling but also shields the machine from radio interference that might disrupt processing. The metal also acts as a heat sink.

Accompanying the computer are three manuals — an operations guide, and MS-DOS and BASIC reference guides. Unusually for this type of machine the guides are paperback books, instead of the more usual ringbinders. The books are held with the system disks in 'mock suede' plastic folders. Of the three manuals, only the operators' guide is tutorially based. Thus anyone wishing to be taken through the various steps of using BASIC or MS-DOS would be advised to buy other instruction books.

As one of the older IBM-compatible machines the Compaq, like the PC itself, uses the Intel 8088 processor instead of the more advanced 8086 chip used by some of its newer rivals such as the Olivetti M25. Although the 8088 is a 16-bit processor with a 20-bit address bus, it has an eight-bit data bus as opposed to the 16-bit bus used in the 8086. This means that the Compaq is much slower in retrieving data than its 8086-equipped competitors, although of course it runs at the same speed as the PC.

On the all-important question of compatibility with IBM software, the Compaq Plus scores highly. Even a notoriously difficult program such as Lotus 1-2-3 will run on the machine. However, as the chairman of Compaq is also on the board of Lotus Software, this is perhaps not surprising. One of the few things that the Compaq Plus will not run is the IBM diagnostic disk, but as that program interrogates the BASIC Input-Output System (BIOS) ROM directly, this should not raise any eyebrows either.

The Compaq Plus is definitely one of the most reliable IBM compatibles on the market, with a proven track record. This counts for a lot to someone who is going to spend £2,524 on a compatible machine. Obviously no one wants to spend that kind of money only to find that a particular program required by the user will not run.

On the negative side, the machine is beginning to show its age — and not only because of the outdated 8088 chip. It is beginning to look as though the days of the 10kg-plus 'transportable' are numbered. With lap-held computers becoming ever more sophisticated, the lap-held IBM-compatible machine is inevitable and 'transportables' such as the Compaq Plus will become redundant.

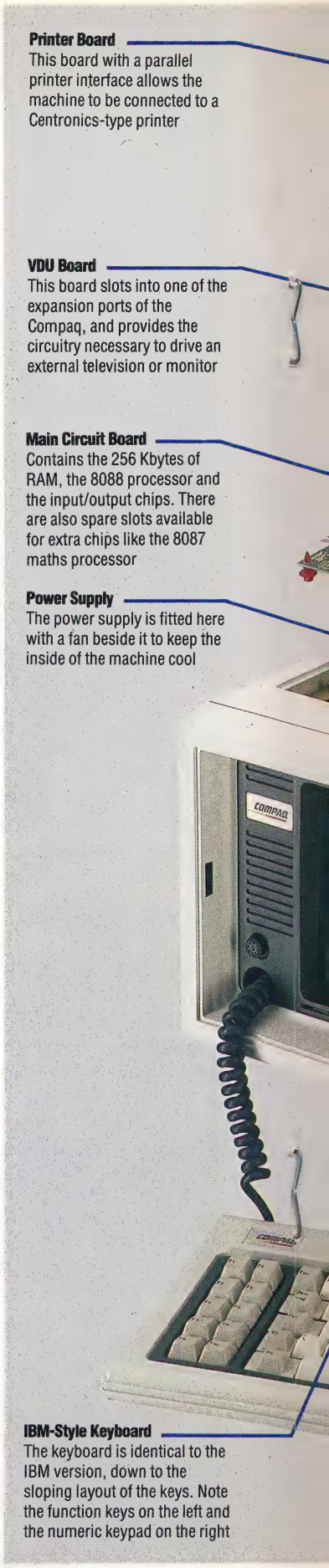
Value For Money

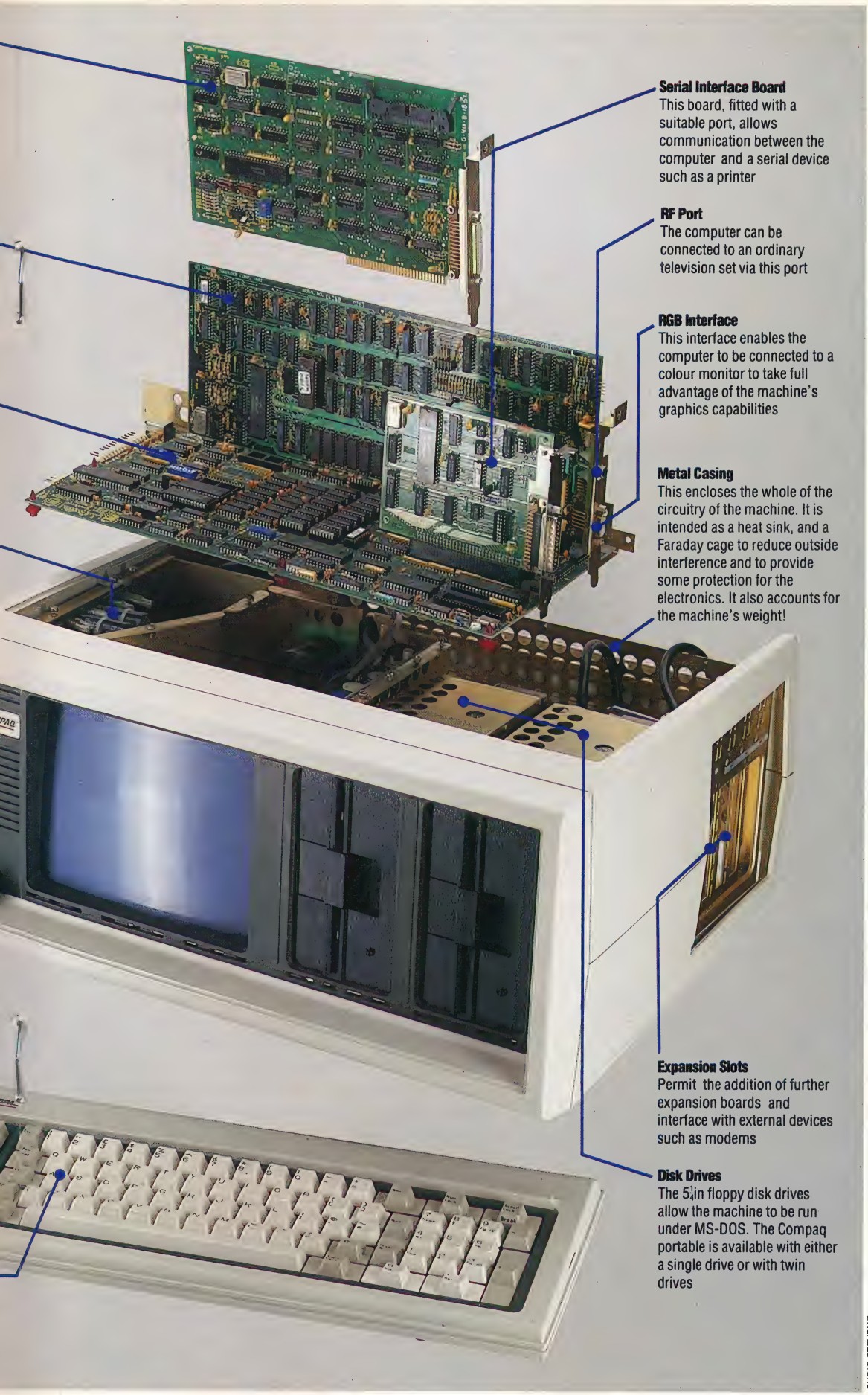
The main selling points of the Compaq Plus are its IBM compatibility and, therefore, access to the enormous IBM software base. Here we show the cost of an IBM PC system that matches the basic Compaq Plus specification:

Feature	Compaq Plus	IBM PC
Basic price	£2,524	£1,310
Twin 360K disks	Standard	Extra £341
Hi-res colour monitor	*Standard	Extra £300
Monitor output card	Standard	Extra £208
Colour graphics card	Standard	Extra £223
128K RAM	Standard	Extra £86
Disk-based operating system and BASIC	Standard	Extra £61
Keyboard	Standard	Extra £213
Total	£2,524	£2,742

*The Compaq's built-in monitor is hi-res monochrome, but the machine has RGB colour monitor output as standard.

CHRIS STEVENS





Serial Interface Board
This board, fitted with a suitable port, allows communication between the computer and a serial device such as a printer

RF Port
The computer can be connected to an ordinary television set via this port

RGB Interface
This interface enables the computer to be connected to a colour monitor to take full advantage of the machine's graphics capabilities

Metal Casing
This encloses the whole of the circuitry of the machine. It is intended as a heat sink, and a Faraday cage to reduce outside interference and to provide some protection for the electronics. It also accounts for the machine's weight!

Expansion Slots
Permit the addition of further expansion boards and interface with external devices such as modems

Disk Drives
The 5¼in floppy disk drives allow the machine to be run under MS-DOS. The Compaq portable is available with either a single drive or with twin drives

CHRIS STEVENS

COMPAQ PLUS

PRICE
£2,524 for twin disk, £2,064 for single disk

DIMENSIONS
480×400×200mm

CPU
Intel 8088,4.7 MHz

MEMORY
256K RAM, expandable to 640K

SCREEN
Text: 25 rows of 80 columns. Graphics: 640×200 pixels

INTERFACES
Centronics type parallel printer, RGB and an RF jack. There are also expansion slots enabling other interface boards to be fitted

LANGUAGES
BASIC loaded from the system master disk

KEYBOARD
47 typewriter-style keys, 14 control keys, 10 function keys and 10 calculator function keys

DOCUMENTATION
Three manuals are provided: an operation guide, MS-DOS reference guide and BASIC reference guide. Of these, only the operations guide takes you step by step through the procedures. Beginners may find that they will need to purchase further manuals for tutorial purposes

STRENGTHS
The machine is sturdily built and has proven compatibility with the IBM PC

WEAKNESSES
The weight of the machine puts it more in the 'transportable' class. The Compaq Plus is also in danger of being superseded by newer technology

INTEGRATED CIRCUIT

An *integrated circuit* (IC) is one that is etched onto a single silicon chip, and performs a specific task. The development of the integrated circuit has revolutionised the electronics industry. ICs have various advantages over discrete devices on a printed circuit board; fundamentally, the miniaturisation that is achieved makes possible an increase in the speed of operation and a reduction in power consumption. Furthermore, once the chip has been designed and developed, manufacturing costs are drastically reduced.

The first integrated circuits were developed in the late 1950s by Jack Kilby, an employee of Texas Instruments. Since then, four generations of integrated circuits have been developed. These are: small scale integration (SSI), medium scale integration (MSI), large scale integration (LSI) and very large scale integration (VLSI). Many scientists now estimate that the limits in the refinement of such miniaturisation have probably been reached and the search is on for a replacement technology.

There are essentially two types of integrated circuit. *MOS* (Metal Oxide Semiconductor) integrated circuits are the more commonly used in microcomputers, since they can be highly packed with the necessary diodes and transistors, and they consume very little power. They also have the advantage of being fairly easy to manufacture.

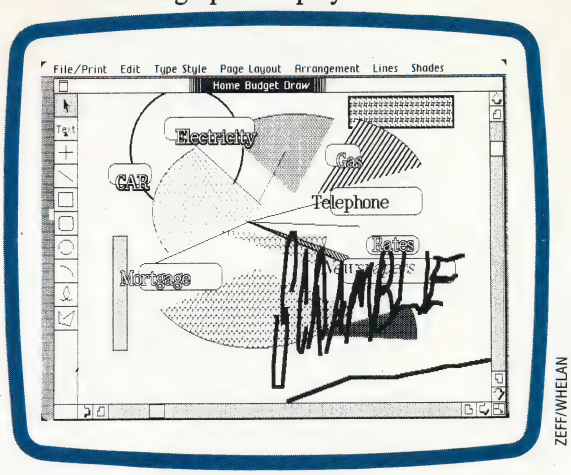
The other type, *bipolar* integrated circuits, are more commonly used on mainframes and minicomputers. These are constructed by the more traditional technology of using positive and negative junction semiconductors to etch the circuitry on the layers of silicon that make up the chip. This makes bipolar ICs more difficult to construct. Other disadvantages are that they have a much lower packing density, and they consume much more power. However, their operating speeds are faster than MOS circuits.

INTELLIGENT TERMINAL

An *intelligent terminal*, usually consisting of a keyboard and screen, is one that is able to perform its own processing — the results of which can then be transferred to a larger computing network, such as a mainframe. Intelligent terminals have a microprocessor onboard to perform tasks such as simple calculations and screen editing — tasks which would otherwise occupy the valuable computing time of the mainframe computer.

INTERACTIVE GRAPHICS

A computer system in which pictures and graphics can be altered and amended in immediate response to an input by the user is said to have *interactive graphics*. The use of such systems is becoming increasingly widespread on microcomputers, particularly with the introduction of integrated software packages such as Lotus 1-2-3, which allows information in a spreadsheet to be displayed instantly as pie, line or bar graphs. In these applications, interactive graphics are used as an aid to understanding the data. Interactive graphics also encompasses such areas as computer aided design (see page 235) and the creation of graphic displays.



ZEFF/WHELAN

Interactive Graphics
Lisadraw is an interactive graphics program for Apple's Lisa computer. Each of the pixels on the Lisa's screen can be turned on or off, giving the user a graphics screen with resolution as high as some CAD/CAM computers

INTERFACE

An *interface* is a hardware device or a software program that provides a boundary between systems or programs, allowing them to pass data. Almost by definition computer systems use signals or data that cannot be understood by other systems. An interface therefore, will contain some method of translating the information into a form that can be understood by the target system.

A hardware interface consists of the cables, plugs and circuitry needed to connect devices together. A software interface is the part of the program that connects mutually exclusive modules or code together. Generally a software interface will pass parameters and variables between one part of the program and another.

INTERPRETER

The central processor units of computers operate in logical fashion on the binary patterns of current that swirl through the system; the people who use computers like to use the words and ideas and symbols that comprise thought. This antithesis is resolved in the programming language, but that still has to be translated for machine consumption. *Interpreters* — machine code programs that translate higher-level languages — are cheap, frugal and effective translation devices — popular with microcomputer manufacturers for those very reasons. When the source program is entered into the system the interpreter intercepts the command keywords and translates them into command codes; otherwise the program is left untouched. When the program is executed the interpreter can act on the keyword codes directly but must translate the rest of the program text by rather clumsy methods; when a program line is executed only action is produced — there is no stored machine code output. This means that in a loop, for example, the interpreter may interpret and execute a line and then repeat this process immediately afterwards as if the instruction had never been encountered before.

CHART TOPPER

Our series of articles on the LOGO language has concentrated on developing routines and procedures for handling data and producing turtle graphics displays. Here we use the language to display data in an easily-understandable form, and we present a routine for the construction of barcharts.

Barcharts are a simple graphical method of representing certain types of numerical data. The aim of a barchart is to help the reader to understand at a glance a set of figures without having to examine them in great detail. Business graphics programs are widely available for plotting barcharts, pie charts, and histograms, and these programs are often linked to spreadsheets so that the values calculated in these can be effectively displayed.

We won't be quite so ambitious for the moment, but let's start by looking at how to draw a barchart using LOGO. The version of the program we will discuss is for the Commodore 64 — see 'Logo Flavours' for any changes that are needed for the program to run on other machines.

The process of drawing a barchart splits up into three stages:

a) get the input;
b) find the largest value (this is necessary so that we can scale the columns to fit on the screen);
c) draw out the barchart, scaling as appropriate, and then add labels.

The top-level procedure is called BARCHART. INIT sets the value of a number of constants needed by the program. By collecting them together in one place, modifications are easier to make. COLORS contains the list of colours to be used for the columns — if you do not like our colour scheme, change it! The program as it stands will print a maximum of 15 columns, so you will need only 15 colours at most.

INPUT AND CALCULATION

The necessary input consists of two data items for each bar of the chart: first, the title to be printed at the foot of the bar, and, second, the quantity or value of the bar — effectively, its height. In our input routines we have incorporated some simple 'validation' checks to make sure that the input is sensible. GET.INPUT splits the job of obtaining input into two parts, getting titles for each column and inserting the relevant values. When you've finished inputting the data, type END as the next 'title'.

GET.TITLE gets the title; this will reject a blank entry but will accept any other value. GET.QUANTITY will accept only a number as an input — any other input will cause the program to ask for the data to be entered again. When GET.INPUT has a valid name/number pair, these are added to the end of a list of data items. Items should be entered in the order you wish to see them plotted across the screen, from left to right.

CALCULATE uses GET.MAX to find the largest value and then uses this to establish a scale. A common rule is that the height of a barchart should be about three-quarters of the width.

DRAWING THE CHART

DRAW.CHART calculates the width of each bar of the barchart, draws an axis up the screen, colours in the bars and then labels them.

The width is calculated as a multiple of eight. This is done in order to avoid some problems caused by the way the Commodore 64 displays colours, for in the normal LOGO graphics mode you can't have two colours in the same eight by eight pixel block. DRAW.AXIS draws the line up the screen and marks on it the highest of the data values; this gives us a simple means of estimating the values of the columns.

We must step back from the axis line before printing this number next to the mask indicating the highest value. How far we need to step back so that the number doesn't overprint the line will depend on how many digits there are in the number. This problem is easily solved, for LOGO treats numbers as if they were words, so we can use COUNT to determine the length of the number, and then use this to determine how far to step back.

WRITE prints a message on the graphics screen. This takes three inputs: the x and y step distances for each character and the name to be written. It uses a primitive, STAMPCHAR, which prints a character on the graphics screen at the turtle's position.

DRAW.CHART1 performs the task of drawing the bars. It takes each item in turn, selects the next colour to be used, scales the height and passes the real work over to FILL, which simply goes up and down the bar filling it in. LABEL uses WRITE to write the labels vertically down the screen (very long labels will 'wrap around' and appear at the top of the screen).

PIE CHARTS

Pie charts are another common form of graphical representation. If you want to write a program to draw a pie chart, here's a few hints:

•Data is obtained in exactly the same way as for the barchart.

•The calculation section involves totalling the numbers and hence determining what share of the 360° making up the 'pie' segment will have.
•Drawing and filling in the slices of the pie is simple enough, but on the Commodore 64, at least, you will have some problems because of the way the colours run into each other. It's a good idea to use 'double colour' mode — just use DOUBLECOLOR instead of DRAW in the procedure that draws the chart. If you are still having problems then leave a 'hole' of 10 units in the centre of the pie chart.

The actual labelling can be done in the same way as for the barchart, although problems may arise when you position the turtle before doing the writing.

Once you've achieved this, why not try writing a program that draws both a pie chart and a barchart for the same data side by side?

Logo Flavours

Many LOGO versions do not have an equivalent to STAMPCHAR, which makes printing characters on the graphics screen very difficult.
The colour numbers and the size of the barchart will need to be altered to suit different machines. All of these details are gathered together in INIT, so that they can be dealt with at once.

For all LCSI versions:
Use TYPE for PRINT1
Use EMPTYP for EMPTY?
You may need to use EQUALP in place of the equals sign (=)
SETXY must be followed by a list
IF has a different syntax — e.g:
IF EMPTYP :DATALIST [STOP]

Good Graph !

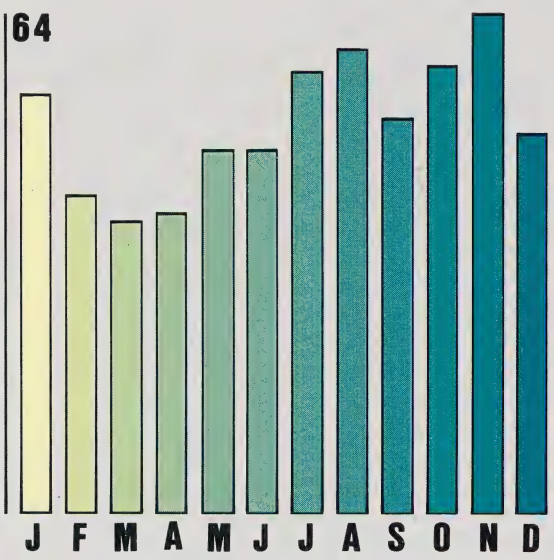
Both barcharts and pie charts are useful ways of displaying information graphically, though LOGO is not really ideal for the purpose because of its slow speed

DATA INPUT

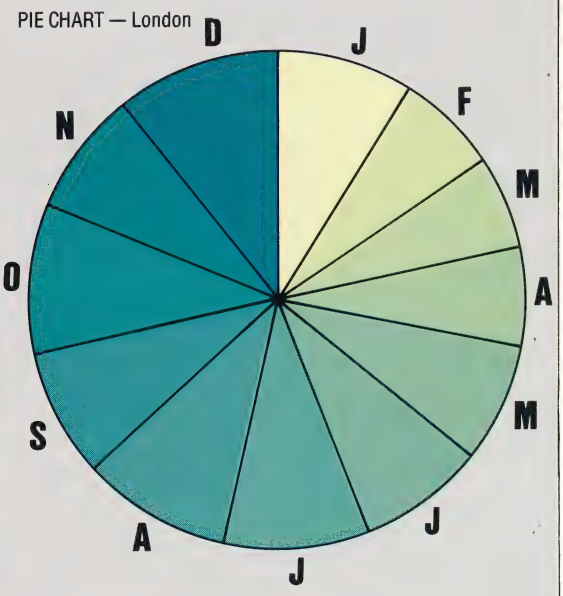
	London	New York
JAN	53	94
FEB	40	97
MAR	37	91
APR	38	81
MAY	46	81
JUN	46	84
JUL	56	107
AUG	59	109
SEP	50	86
OCT	57	89
NOV	64	76
DEC	48	91

Rainfall Data (in millimetres)

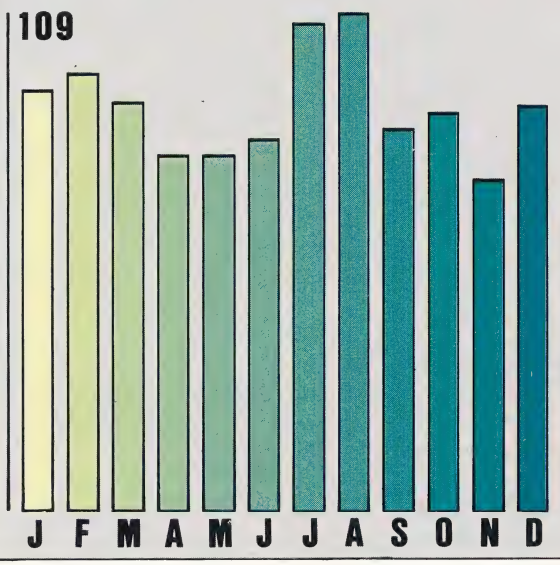
BARCHART — London



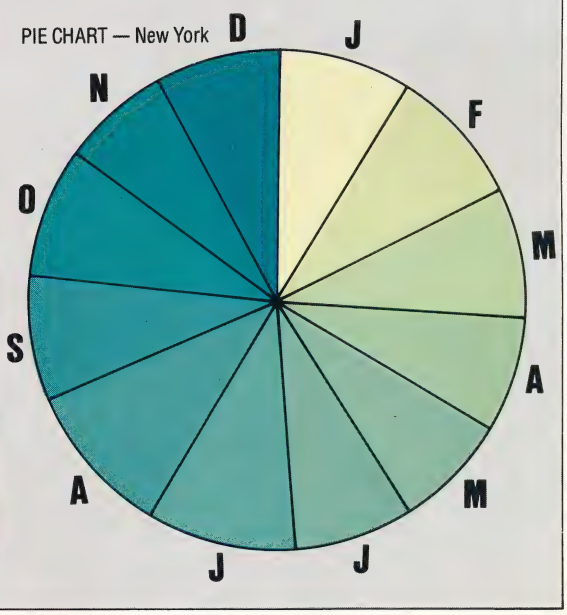
PIE CHART — London



BARCHART — New York



PIE CHART — New York



The Barchart Program

```
TO BARCHART
  INIT NODRAW
  GET.INPUT CALCULATE
  DRAW.CHART
END

TO INIT
  MAKE "COLORS [0 1 2 5 6 0 1 2 5 6 0 1 2 5 6]
  MAKE "BACKG 11
  MAKE "DATA []
  MAKE "HEIGHT 160
  MAKE "WIDE 190
  MAKE "XSTART ( – 104 )
  MAKE "YSTART ( – 40 )
END

TO GET.INPUT
  GET.TITLE
  IF FIRST :TITLE = "END THEN STOP
  GET.QUANTITY
  MAKE "DATA LPUT SENTENCE :TITLE :QUANTITY
    :DATA
  GET.INPUT
END

TO GET.TITLE
  PRINT "
  PRINT [ (TYPE END TO STOP) ]
  ( PRINT1 "TITLE: "' ' )
  MAKE "TITLE REQUEST
  IF EMPTY? :TITLE THEN GET.TITLE
END

TO GET.QUANTITY
  ( PRINT1 "QUANTITY: "' ' )
  MAKE "QUANTITY REQUEST
  IF EMPTY? :QUANTITY THEN GET.QUANTITY
    ELSE IF NOT NUMBER?
  FIRST :QUANTITY THEN GET.QUANTITY
END

TO CALCULATE
  MAKE "MAX 0
  GET.MAX :DATA
  MAKE "SCALE :HEIGHT / :MAX
END

TO GET.MAX :DATALIST
  IF EMPTY? :DATALIST THEN STOP
  IF LAST FIRST :DATALIST > :MAX THEN MAKE
    "MAX LAST FIRST
    :DATALIST
  GET.MAX BUTFIRST :DATALIST
END

TO DRAW.CHART
  DRAW BACKGROUND :BACKG
  HIDETURTLE FULLSCREEN
  MAKE "WIDTH ( ( ROUND ( ( :WIDE / COUNT
    :DATA ) / 8 ) ) * 8 ) – 8
  PENUP SETXY :XSTART :YSTART
  DRAW.AXIS
  DRAW.CHART1 :COLORS :DATA
  LABEL :DATA
END

TO DRAW.AXIS
  PENDOWN
  SETY YCOR + :HEIGHT + 10
  SETY YCOR – 10
  SETX XCOR + 4
  SETX XCOR – 8
  PENUP
  SETX XCOR – ( 8 * COUNT :MAX )
  WRITE 8 0 :MAX
  SETX XCOR + 4
  SETY :START
END

TO WRITE :XINC :YINC :NAME
  IF EMPTY? :NAME THEN STOP
  STAMPCHAR FIRST :NAME
  SETXY XCOR + :XINC YCOR + :YINC
  WRITE :XINC :YINC BUTFIRST :NAME
END

TO DRAW.CHART1 :PENCOLORS :DATALIST
  IF EMPTY? :DATALIST THEN STOP
  SETX XCOR + 8
  PENCOLOR FIRST :PENCOLORS
  PENDOWN
  FILL ( ( LAST FIRST :DATALIST ) * :SCALE )
    :WIDTH
  PENUP
  DRAW.CHART1 BUTFIRST :PENCOLORS
    BUTFIRST :DATALIST
END

TO FILL :LEN :WID
  IF :WID = 0 THEN STOP
  FORWARD :LEN RIGHT 90
  FORWARD 1 LEFT 90
  BACK :LEN RIGHT 90
  FORWARD 1 LEFT 90
  FILL :LEN :WID – 2
END

TO LABEL :DATALIST
  LABEL1 ( :XSTART + 8 + :WIDTH / 2 ) ( :YSTART
    – 10 )
    :DATALIST
END

TO LABEL1 :X :Y :DATALIST
  IF EMPTY? :DATALIST THEN STOP
  SETXY :X :Y
  WRITE 0 ( – 10 ) FIRST FIRST :DATALIST
  LABEL1 ( :X + :WIDTH + 8 ) :Y BUTFIRST
    :DATALIST
END
```

MICRO ELECTRONICS

In the last instalment of Workshop, we completed the first phase of assembly of our robot and tested this by writing a short program to bring it under keyboard control. Now we mount four microswitch sensors on the robot and write a simple program to test their operation.

Bi-directional control of the stepper motors requires four of the eight user port data lines available to us. This leaves four lines that can be used to carry information from the sensors back to the computer. To give our robot more flexibility in its operation, we will use a 'patching system' to allow the connection of different permutations of the sensors to the four available input lines. For the moment, we will connect four microswitch sensors and in the future we will instal two light sensors. So that we can select any combination of these sensors — for example, two microswitch sensors and two light sensors — we will wire each sensor to a socket on the lid of the robot. Four sockets will also be connected to the data lines — D4 to D7 — on the D plug. We can therefore connect the appropriate sensor to any of the four data lines by using a short patch lead, which plugs into the sensor socket at one end and one of the data line sockets at the other.

To test the construction and wiring of the four microswitches we can write a very simple program that scans the upper four bits of the data register and displays the decimal values of the bits sent low. Run the program with all four sensors connected, via the patch socket system, to the data lines, D4 to D7. Closing any of the microswitches will cause the screen display to change.

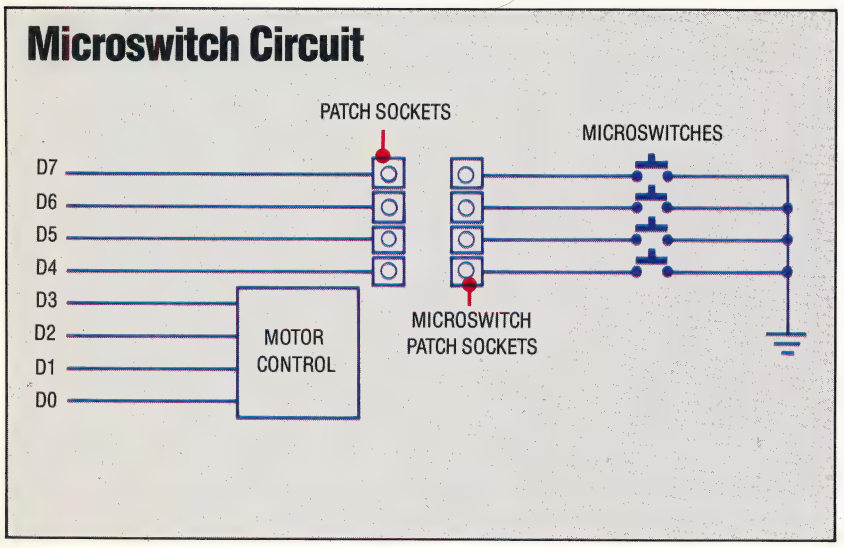
Fitting The Microswitches

Maplins do not stock suitable microswitches for our needs, but the sort mentioned in the parts list are available from many other component stockists. First, make the cut-outs required in the robot case lid. The eight slits should be cut so that the plate connectors under each microswitch can just fit through. The 10 5mm-diameter holes accommodate the patch sockets, and these may be fitted at this stage — the four red sockets being mounted in a line closest to the D plug. The microswitches have to be adapted slightly for this project. The long microswitch lever must be carefully bent, using a pair of pliers, so that a right-angle bend is made. Care should be taken to ensure that the bend is not made so close to the microswitch housing that the lever will not close the microswitch when mounted on the lid. Two plate connectors protrude from the back of the microswitch. The upper of these is the NC, or 'normally-closed', connector. As this will not be used in our project, you may wish to remove it, either by breaking it off or sawing it through with a small hacksaw. The lower NO, or 'normally-open', connector is used, and this should be bent, at right-angles, close to the switch case. This will ensure that it will fit, together with the COM, or 'common-connector', through the slots cut for it in the robot case lid. When these adaptations have been made to each of the four microswitches, they can be mounted at each corner of the lid, as shown. The switches should be mounted so that the activating levers hang over the front and rear of the robot case, and they should be glued in place using a suitable adhesive. Cyanoacrylate or 'super glue'-type glues are probably the best to use

```
10 REM **** BBC SENSOR TEST ****
20 MODE 7:OP=-1:DDR=&FE62:DATREG=&FE60:?DDR=15
30 PE=240-(?DATREG AND 240):IF PE=OP THEN 30
40 CLS:PRINT PE:OP=PE:GOTO 30
```

```
10 REM **** CBM SENSOR TEST ****
20 OP=-1:DDR=56579:DATREG=56577:POKE DDR,15
30 PE=240-(PEEK(DATREG) AND 240):IF PE=OP THEN 30
40 PRINT CHR$(147):PRINT PE:OP=PE:GOTO 30
```

Microswitch Circuit

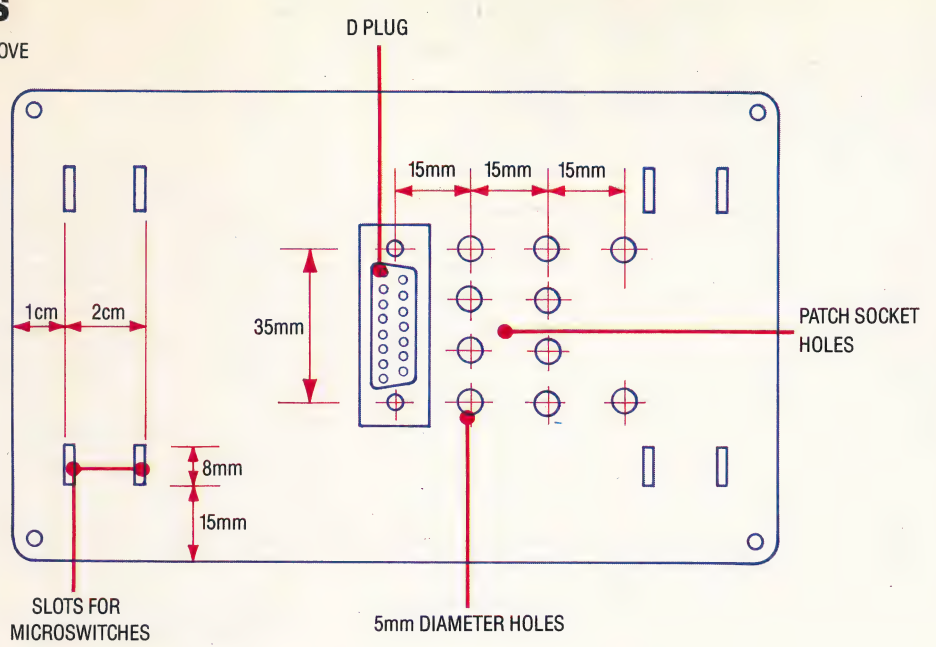


Microswitch Circuit

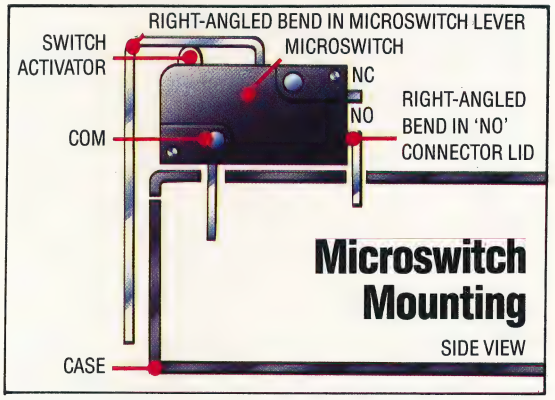
The circuit that connects the microswitch sensors into the robot system is very simple. Data lines D4 to D7 connect to four sockets mounted on the lid of the robot; data lines D0 to D3 are used for motor control. One side of each microswitch connects to a similar group of four sockets; the other side connecting to a common earth. If all four microswitches are required, then they can be patched into the data lines by four patch cords. If the upper four bits of the user port data register are set to input then they are normally held high (to one). Closing any microswitch patched into the system will connect the relevant data line to earth, bringing the corresponding bit in the data register low (to zero)

Lid Holes

VIEWED FROM ABOVE



KEVIN JONES



Microswitch Mounting

Parts List

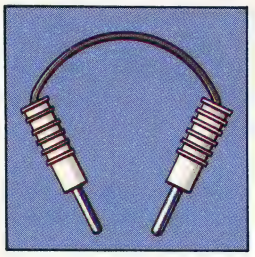
MAPLIN

No	Item	Source
8	2mm plugs	HF38R
4	2mm red sockets	HF44X
6	2mm blue sockets	HF44X

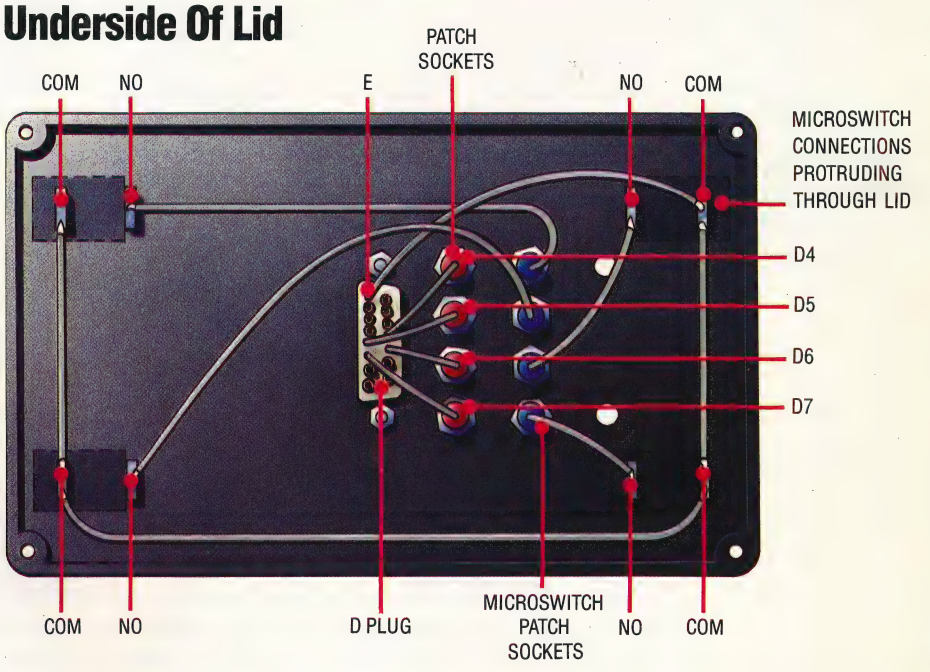
MISCELLANEOUS

4 microswitches Cherry Elec. E22
1m 4-way ribbon cable

With the microswitches mounted, turn the lid over. Solder short lengths of covered wire to each of the four red patch sockets and connect these to the relevant pins on the D plug. The four COM connectors should be connected to each other, and the earth pin on the D plug, in a ring circuit, as shown. Each NO connector should then be connected to the appropriate blue socket. Make up four patch cords for use with the patch system



Underside Of Lid



KEVIN JONES

BBC MEMORY MAPPING

In the first part of this series we looked at how an operating system is arranged in general terms and examined, in particular, how the BBC Micro's OS is arranged. In this instalment, we'll look at how the BBC OS actually uses its memory, and how vectors are used in a computer operating system.

The BBC Micro makes use of various areas of memory, many of which have multiple functions. A typical example of this is page &9 of the memory — the area of RAM between locations &900 and &9FF — which is used, at different times, as the RS232 output buffer, a cassette output buffer, speech workspace and extended sound workspace! No one can accuse Acorn of not getting the most from its memory.

A further example is the block from &0E00 to &1900; in a BBC Micro with a cassette recorder as its filing system, this area of memory is free for use as BASIC program space. However, when a disk system is being used, this area is used as disk filing system workspace, reducing the amount of memory left for BASIC programs by over 2.5 Kbytes. This is extremely annoying in screen Modes 0 to 2, where memory is restricted anyway.

The final example of multiple use of memory is the area between &8000 and &BFFF. If a DFS ROM, a word processor or utility ROM is fitted in the machine it will occupy this block. This is also the area of memory that the BASIC interpreter ROM resides in. These are called 'paged ROMs'; only one of them can be active at any one time. The ROM required is selected by the operating system, and 'switched in', or 'paged in' — hence the name. Under normal circumstances, the BASIC ROM is paged in, and the BASIC interpreter is running, thus enabling us to type in and execute BASIC programs. However, when we execute a DFS command, the OS pages the DFS ROM, executes the command, and then pages the BASIC ROM back in. During the execution of the DFS commands, the BASIC ROM is simply ignored. Other ROMs, such as the Telesoftware Filing System or the Wordwise word processor chip, also occupy this memory space and are also paged in and out as required. It's probably just as well that the BBC Micro uses the same area of memory for several purposes, as otherwise there'd be little memory left for us to work with.

USER MEMORY

Once the OS has taken its share of the computer memory, what remains is the available user memory. The BASIC system variable PAGE contains the address of the Start of BASIC program area, and the variable HIMEM points to the start of the screen display memory; the space between is available for BASIC programs, variables and machine code programs. Type in this instruction to find these addresses, and the memory available for BASIC programs:

```
PRINT~PAGE,~HIMEM ~(HIMEM-PAGE) "BYTES FREE"
```

In the following table, you'll find a brief description of the relative merits of different storage areas for machine code in the BBC Micro.

When Area is Suitable For Machine Code Programs	Type of Filing System	
	Tape	Disk
&0D00 — &0DFF Page &D—only if no paged ROMs are in the machine	(✓)	X
&0B00 — &0CFF Page &B and &C — if no user defined characters or user function keys are used in the program	(✓)	(✓)
&0A00 — &0AFF Page &A — if serial interface is not in use	(✓)	(✓)
Space saved in BASIC program area in the DIM command		

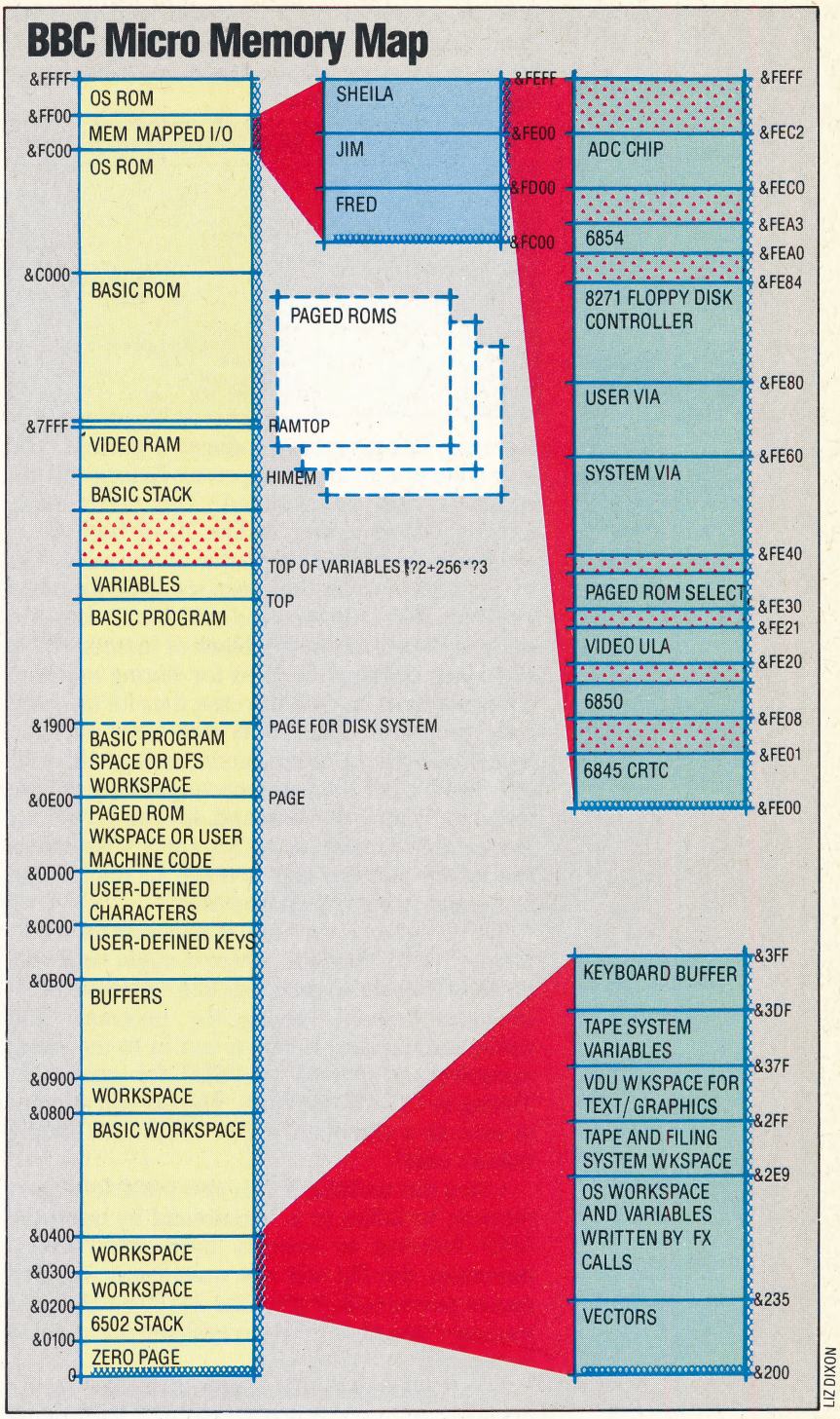
✓= OK (✓) = OK, with Reservations X Not used

For most routines that are to be used in conjunction with BASIC programs, storing the code in the BASIC program area is the best method.

Once the machine code is in memory, it generally needs some workspace to allow it to run. Most 6502 machine code programs require some zero page locations as workspace, mainly because certain indexed addressing instructions will need zero page locations. Acorn have made extensive use of this page for the OS, but some locations have been set aside for machine code programming. A byte by byte account of what each of the OS locations does is little use to the programmer, and directly accessing these locations is not encouraged — the useful locations should be accessed by calling the appropriate OS versions, thus creating problems when a machine code program written under one OS version is run under a different version.

ENTERING THE OS

The two major routes we can take to use the routines resident in the BBC OS are the OSBYTE and OSWORD routines:



● OSBYTE enables us to affect the behaviour of many OS routines, by passing control codes and parameters in the A, X and Y registers of the 6502. In BASIC, we can access the OSBYTE routines via the *FX command. *FX is followed by either two or three numbers: the first number is the control or function code passed to the A register; the second is the number passed to the X register; and the third is the number passed to the Y register. The third parameter is not required by all OSBYTE calls. In machine code, OSBYTE is called at address &FFF4. These two versions are equivalent in function:

BASIC	Assembly Language
*FX4,1	LDA #4 LDX #1 JSR &FFF4

The value in the A register defines exactly what a particular call to OSBYTE will do. The above call, for example, affects the actions of the cursor keys; the parameter passed over in the X register specifies whether the cursor keys retain their normal editing function or whether they simply return an ASCII code.

It's a sad fact of life that all OS routines cannot be affected by OSBYTE. Its main drawback is the limit on the number of parameters that you can pass over to the routine. If we ignore the contents of the A register, which tells the OS which routine within OSBYTE we wish to use, then we can only pass two parameters in the X and Y registers. If we want to pass any more than this then we use the second routine, OSWORD.

● OSWORD enables us to do such things as sound generation, disk reads and writes, and so on. This is the difference between OSWORD and OSBYTE; OSBYTE affects *how* the OS does certain tasks, and OSWORD enables us to *do* particular tasks. OSWORD obtains its parameters from a *parameter block* that is pointed to on entry to the OSWORD routine by the 6502 X and Y registers. This parameter block is situated in RAM, and its size and arrangement depend upon the OSWORD call being made. The A register contains a function code that determines which of the OSWORD functions are to be executed by the OS. Once the registers and the parameter block have been prepared, OSWORD is called at address &FFF1. OSBYTE and OSWORD are the principal means of entry to the OS; because of their importance, we'll look at them in greater detail in later parts of the series.

Other OS routines are entered, from BASIC, by typing in an asterisk (*) followed by the command. The presence of the * causes the command following it to avoid the BASIC interpreter and be passed to an OS routine that bears the name OSCLI, meaning the 'Operating System Command Line Interpreter'. This interprets the OS commands that are typed in and acts upon them by calling the appropriate routines in the OS. Such commands are often called * or Star Commands. The table shown lists those Star Commands recognised by the BBC; any not recognised, spelt incorrectly or without the correct number of parameters will usually give the Bad Command error message.

We can pass commands to the Command Line Interpreter (CLI) by using the OS routine or by the direct method. OSCLI's uses are twofold: first of all, it enables us to pass the commands shown in the table to the CLI from machine code should we want to; and secondly it enables us to pass BASIC string variables over to the CLI. The programs that follow feature both these uses. Notice that the integer variables, A%, X% and Y%, pass their values directly into the A, X and Y registers. X% (or the X register) and Y% (or the Y register) point to the position in memory of the string of characters, that

is to be treated as a * command, and is to be interpreted and executed by the OSCLI routine. X holds the low byte of the address and Y holds the high byte of the address.

BASIC Version

```
10 DIM C 100
20 OSCLI=&FFF7
30 INPUT"ENTER COMMAND
",A$
40 $C=A$
50 X%=C MOD 256
60 Y%=C DIV 256
70 CALL OSCLI
80 GOTO 30
```

BASIC+Assembly Language Version

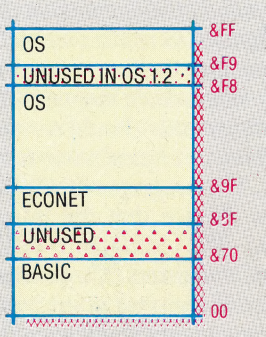
```
 10 DIM C 100
 20 OSCLI=&FFF7
 30 FOR I%=0 TO 2 STEP 2
 40 P%=&C00
 50 [ OPT I%
 60 .code    LDX #C MOD 256
 70          LDY #C DIV 256
 80          JSR OSCLI
 90          RTS
100 ]:NEXT I%
110 INPUT"ENTER COMMAND ",A$
120 $C=A$
130 CALL code
140 GOTO 110
```

Running this program produces a prompt: the user types in a * command, presses Return and the command will be executed. The rather odd-looking DIM statement in line 10 of both these programs forces the computer to set aside 100 bytes of memory in the space set aside for BASIC variables and initialises the variable C with the address of the start of this block of memory. This 100 bytes can then be used for storing machine code programs, or, as in this case, data for machine code programs. The $C=A$ statement puts the bytes that make up the command string held in A$ into the block of 100 bytes, starting at the first byte reserved by the DIM command. In both programs, the X and Y registers (or the X% and Y% variables) are set up and the call is made to OSCLI. The command string is then executed.

This program is the basis of a routine for use in menu-driven programs, where it might be useful to allow the user to do things like catalogue disks or tapes without leaving the program. The command required is simply put in to the string variables and passed to OSCLI for execution. Typing in *A$ will not work. The OS will attempt to execute a command called *A$, which simply doesn't exist!

Using this technique, it is also possible to pass numeric variables to a * command by using the STR$ function to convert them into strings. Normally, the CLI will not accept any variable names passed to it; it gives the Bad Command error message instead of trying to evaluate the variable concerned.

Any * command that is not recognised by the OS is passed over to any paged ROMs. Each one is asked if it recognises the command; if it does then it executes it. Commands not passed for execution in this fashion are then treated as bad commands *only if* a fast filing system, such as a disk drive, is not in use. If it is, then the disk will be inspected to see if it contains a file with the same name as the command (without the *). If it does, then the file is loaded into the machine and treated as a machine code program. This can cause *big* problems if the file isn't a machine code program! The computer usually 'hangs' until you put it out of its misery. If such a file isn't found then the Bad Command error message is printed.



BBC Micro Zero Page Usage
The 6502's limited indexing facilities are complemented by its view of the Zero Page (&0000 to &00FF) as an array of auxiliary index registers. This naturally leads to heavy use of these locations by the OS, but a few bytes have been left unused for programmers' purposes

The * Commands

Command	Description and Comments
*HELP	This will give the version number of the BASIC. It can also be used to gain information about other paged ROMs fitted — e.g. *HELP DFS
*BASIC	This command will enter the BASIC language. A variation of this command, such as *WORD, will enter a paged ROM, in this case View
*CODE *LINE	In OS 1.2 only. This enables the user to add new commands to the OS
*KEY	Used to program the red function keys
*MOTOR n	Used to control the tape motor relay: n=0 turns the relay off and n=1 turns the relay on
*ROM, *TAPE, *DISK, *NET	Used to initialise the appropriate filing system — i.e. *TAPE will enter the 1,200 baud tape filing system, and *DISK will enter the disk filing system
*FX	This enables the programmer to control the values of various OS variables, thus controlling the behaviour of the OS
*RUN, *OPT *LOAD, *. *CAT, *SAVE	These are all filing system commands and will be examined later in the series
*SPOOL *EXEC	*SPOOL sends the screen output to the screen and to a file. *EXEC reads in data from a file as if it were being read from the keyboard
*TV x,y	Affects the vertical position of the screen and the screen interface: x=0 means no change in vertical position, x=1 moves screen down one line, x=255 moves screen up one line; y=0 interlace on, y=1 interlace off. The effects of this command come into operation at the next mode change and stay in effect until a CTRL-BREAK or another *TV command is issued

For further explanation of the use of these commands, consult the BBC Micro's user guide

NOW IS THE TIME TO ORDER ANY COPIES OR BINDERS YOU MAY BE MISSING FROM YOUR COLLECTION.

Copies of any weekly issue can be obtained, subject to the availability of stocks, by using this reply-paid order form and marking clearly which issues you require to be sent to you.

Each issue costs 80 pence including postage and packing, and please enclose your cheque/postal order made payable to Orbis Publishing Limited.

Back Numbers Order Form.

Please send me the back numbers I have circled below.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43					

Each issue costs 80 pence including postage and packing.

Binder Order Form.

Please send me the binder volume numbers I have ticked below.

☐ Volume 1 ☐ Volume 2
☐ Volume 3 ☐ Volume 4

Binders are £3.95 each inclusive of postage and packing.

Important: Please read this carefully:

1. Do not complete this order form if you have already asked for binders to be sent to you automatically as they are issued.

2. Readers not in the UK or The Republic of Ireland, see inside front cover for details of how to obtain binders and back numbers.

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd, for a total of £_____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of both back numbers and binders.

When you have completed the order form above for back numbers and/or binders, fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully.

NO STAMP NECESSARY.

BLOCK CAPITALS PLEASE

Complete the section below with one letter, figure or space per square.

Mr
Mrs
Miss

Initials

Surname

Address

Send this page with your cheque/postal order to:
The Home Computer Advanced Course,
Orbis Publishing Limited,
Freepost
Orbis House, 20-22 Bedfordbury,
London WC2N 4BR.

(No stamp necessary)